# TutorialsPoint Java 技术教程

wizardforcel

Published with GitBook



# 目錄

# TutorialsPoint Java 技术教程

# Apache ANT 教程

Apache Ant是由Apache软件基金会一个基于Java的构建工具。Apache Ant的构建文件是用XML编写，并采取了开放的标准，便于携带和易于理解的XML性质的优势。

本教程将教你如何使用Apache Ant 自动构建和部署过程。

## 读者

本教程是专为初学者，帮助他们了解了Apache Ant工具来自动构建和部署过程的基本功能。

## 必备要求

我们假设你有软件开发知识使用编程语言，Java编程和软件构建和部署过程。

# ANT简介 - ANT

## 为什么你需要一个构建工具？

理解Apache Ant定义之前，必须了解需要一个构建工具。为什么我需要Ant，或者更具体地说，为什么我需要一个构建工具？

花你一天做以下工作？

- 编译代码
- 打包二进制文件
- 部署二进制文件到测试服务器
- 测试您的代码更改
- 从一个位置复制代码到另一个地方

如果你回答是肯定的上述任何一项，那么现在是时候实现过程的自动化。

平均而言，开发人员花费3小时（工作日超出 8小时）做这样构建和部署平凡的任务。难道你会很高兴多要回3个小时？

Apache Ant是可以在命令行中执行一个操作系统构建和部署工具。

## Apache Ant的历史

- Ant 代表着另一种简洁的工具
- Ant 是由詹姆斯·邓肯·戴维森（Tomcat的原作者）创建的，在他欧洲飞往美国时。
- Ant 最初是用来构建Tomcat，被捆绑Tomcat作为分发的一部分
- Ant 诞生制造工具出有关的问题和复杂性
- Ant 于2000年晋升为在Apache的一个独立项目。
- Apache Ant（截至2011年7月）的当前版本是1.8.2
- NAnt 是.NET构建工具，它类似于Ant，但用于构建.NET应用程序

## Apache Ant功能

- Ant 是最完整的Java构建和部署工具。

- Ant是平台无关的，可以处理特定平台的属性，如文件分隔符。
- Ant 可以用于执行特定任务的平台，例如使用“触摸'命令修改文件的修改时间。
- Ant 脚本使用的是纯XML编写的。如果你已经熟悉XML，你可以学习Ant 很快。
- Ant擅长复杂的自动化重复的任务。
- Ant 自带的预定义任务的大名单。
- Ant提供了开发自定义任务的界面。
- Ant可以在命令行中很容易地调用，它可以与免费的和商业的IDE集成。

# ANT环境安装设置 - ANT

## 安装Apache Ant

假定您已经下载并安装Java开发工具包（JDK）在您的电脑上。如果没有，请按照这里的说明。

Apache Ant是Apache软件许可证，由开放源码倡议认证一个完全成熟的开源许可下发布。

最新的Apache Ant版本，包括完整的源代码，类文件和文档可以在这里找到 http://ant.apache.org.

- 确保将JAVA_HOME环境变量设置到安装JDK的文件夹。
- 下载的二进制文件从http://ant.apache.org
- 使用Winzip，WinRAR，7-zip或类似工具解压缩zip文件到一个方便的位置 c:folder.
- 创建一个名为ANT_HOME，一个新的环境变量指向Ant的安装文件夹，在 c:apache-ant-1.8.2-bin 文件夹。
- 附加的路径Apache Ant批处理文件添加到PATH环境变量中。在我们的例子是 c:apache-ant-1.8.2-binin文件夹。

## 验证Apache Ant的安装

要验证Apache Ant已成功安装在您的计算机上，打开一个命令提示符，然后输入ant。

你应该会看到类似的输出：

```
C:>ant -version
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

如果您看不到上面的，那么请确认您已遵循安装步骤正确。

## 安装 Eclipse

本教程还包括集成Ant 到 Eclipse IDE中。所以，如果你还没有安装Eclipse，请下载并安装Eclipse

想要Eclipse:

- 从下载最新的Eclipse二进制文件 www.eclipse.org
- 解压Eclipse的二进制文件到一个方便的位置如 c:
- 运行 c:eclipseeclipse.exe

# Ant构建文件 - ANT

通常情况下，Ant构建文件build.xml应该在项目的基础目录。可以自由使用其他文件名或将构建文件中其他位置。

在本练习中，创建一个名为build.xml 在电脑的任何地方的文件。

```
<?xml version="1.0"?>
   <project name="Hello World Project" default="info">
   <target name="info">
      <echo>Hello World - Welcome to Apache Ant!</echo>
   </target>
</project>
```

请注意，应该有XML声明之前没有空行或空格。该处理指令目标匹配"[xX][mM][lL]"是不允许的 - 如果你这样做，这可能在运行Ant构建时造成的错误消息。

所有构建文件要求项目元素和至少一个目标元素。

XML元素的项目有三个属性：

| 属性 | 描述 |
|---|---|
| name | The Name of the project. (Optional) |
| default | The default target for the build script. A project may contain any number of targets. This attribute specifies which target should be considered as the default. (Mandatory) |
| basedir | The base directory (or) the root folder for the project. (Optional) |

一个目标是要作为一个单元运行的任务的集合。在我们的例子中，我们有一个简单的目标，以提供一个信息性消息给用户。

目标可以对其他目标的依赖关系。例如，部署目标可能对封装对象的依赖和包的目标可能具有依赖于compile目标等等。依赖关系是使用依赖属性表示。例如：

```
<target name="deploy" depends="pacakge">
  ....
</target>
<target name="pacakge" depends="clean,compile">
  ....
</target>
<target name="clean" >
  ....
</target>
<target name="compile" >
  ....
</target>
```

目标元素具有以下属性：

| 属性 | 描述 |
|---|---|
| name | The name of the target (Required) |
| depends | Comma separated list of all targets that this target depends on. (Optional) |
| description | A short description of the target. (optional) |
| if | Allows the execution of a target based on the trueness of a conditional attribute. (optional) |
| unless | Adds the target to the dependency list of the specified Extension Point. An Extension Point is similar to a target, but it does not have any tasks. (Optional) |

在上面的例子中的echo 任务是打印一条消息一个简单的任务。在我们的例子，它打印出Hello World消息。

要运行Ant构建文件，打开命令提示符并导航到build.xml文件所在的文件夹，然后输入ant info。也可以只输入ant来代替。既会工作，因为信息是默认的目标在构建文件。应该看到下面的输出：

```
 C:>ant
Buildfile: C:uild.xml

info:
     [echo] Hello World - Welcome to Apache Ant!

BUILD SUCCESSFUL
Total time: 0 seconds

C:>
```

# Ant属性任务 - ANT

Ant构建文件是用XML编写的，它不迎合声明变量，你在最喜欢的编程语言做的。然而，正如你可能已经想到，它会如果允许Ant声明变量，如项目名称，项目源代码目录等有用

Ant使用属性元素，它允许你指定的属性。这允许属性从一个版本改变为另一个。或者从一个环境到另一个。

默认情况下，Ant提供了可以在构建文件中使用下列预定义的属性

| 属性 | 描述 |
| --- | --- |
| ant.file | The full location of the build file. |
| ant.version | The version of the Apache Ant installation. |
| basedir | The basedir of the build, as specified in the **basedir** attribute of the**project** element. |
| ant.java.version | The version of the JDK that is used by Ant. |
| ant.project.name | The name of the project, as specified in the **name** atrribute of the**project** element |
| ant.project.default-target | The default target of the current project |
| ant.project.invoked-targets | Comma separated list of the targets that were invoked in the current project |
| ant.core.lib | The full location of the ant jar file |
| ant.home | The home directory of Ant installation |
| ant.library.dir | The home directory for Ant library files - typically ANT_HOME/lib folder. |

Ant也使得系统性能（例如：文件分割符），可用于构建文件。

除了以上所述，用户可以使用属性元素定义附加属性。一个例子介绍如下展示了如何定义一个名为站点名称（sitename）属性：

```
<?xml version="1.0"?>
<project name="Hello World Project" default="info">
  <property name="sitename" value="www.yiibai.com"/>
   <target name="info">
      <echo>Apache Ant version is ${ant.version} - You are
      at ${sitename} </echo>
   </target>
</project>
```

上述构建文件运行ant应该产生下面的输出：

```
C:>ant
Buildfile: C:uild.xml

info:
     [echo] Apache Ant version is Apache Ant(TM) version 1.8.2
     compiled on December 20 2010 - You are at www.yiibai.com

BUILD SUCCESSFUL
Total time: 0 seconds
C:>
```

# Ant属性文件 - ANT

直接在构建文件中设置属性是好的，如果你使用的是少数属性。然而，对于一个大型项目，是要存储在一个单独的属性文件中。

存储在一个单独的文件中的属性可以让你重复使用相同的编译文件，针对不同的执行环境不同的属性设置。例如，生成属性文件可以单独维持DEV，TEST和PROD环境。

指定在一个单独的文件属性是有用的，当你不知道一个属性（在一个特定的环境中）前面的值。这使您可以在属性值是已知的其他环境进行构建。

没有硬性规定，但一般属性文件名为build.properties文件，并放在沿一侧的build.xml文件。如build.properties.dev和build.properties.test - 你可以根据部署环境中创建多个生成属性文件

构建属性文件的内容类似于普通的Java属性文件。他们每行包含一个属性。每个属性由一个名称和一个值对来表示。名称和值对由等号分开。强烈建议属性标注了正确的注释。注释列出所使用的哈希字符。

下面显示了一个build.xml文件和相关build.properties文件

## build.xml

```
<?xml version="1.0"?>
<project name="Hello World Project" default="info">
   <property file="build.properties"/>
      <target name="info">
         <echo>Apache Ant version is ${ant.version} - You are
         at ${sitename} </echo>
     </target>
</project>
```

## build.properties

```
# The Site Name
sitename=www.yiibai.com
buildversion=3.3.2
```

在上面的例子中，站点名是被映射到该站点的名称自定义属性，可以声明任意数目以这种方式自定义属性。在上面的例子中所列的另一个自定义属性是buildversion，其中，在这种情况下指的是创建的版本。除上述者外，Ant附带了一些预定义的构建属性，它已被列入上一节中，但下面是代表一次。

| 属性 | 描述 |
|---|---|
| ant.file | The full location of the build file |
| ant.version | The version of the Apache Ant installation |
| basedir | The basedir of the build, as specified in the **basedir** attribute of the**project** element. |
| ant.java.version | The version of the JDK that is used by Ant. |
| ant.project.name | The name of the project, as specified in the **name** atrribute of the**project** element. |
| ant.project.default-target | The default target of the current project |
| ant.project.invoked-targets | Comma separated list of the targets that were invoked in the current project |
| ant.core.lib | The full location of the ant jar file |
| ant.home | The home directory of Ant installation |
| ant.library.dir | The home directory for Ant library files - typically ANT_HOME/lib folder. |

在本节的例子中，我们使用内置的属性ant.version。

# Ant数据类型 - ANT

Ant提供了一些预定义的数据类型。不要混淆，也可在编程语言中的数据类型，而是考虑数据类型的设置被内置到产品中服务。

下面是一个由Apache Ant的提供的数据类型的列表

## 文件集合

该文件集的数据类型表示文件的集合。该文件集的数据类型通常是作为一个过滤器，以包括和排除匹配特定模式的文件。

例如：

```
<fileset dir="${src}" casesensitive="yes">
  <include name="**/*.java"/>
  <exclude name="**/*Stub*"/>
</fileset>
```

在上面的例子中的src属性指向项目的源文件夹。

在上面的例子中，文件集的选择，除了那些包含在其中单词“Stub”源文件夹中的所有java文件。在大小写敏感的过滤器应用到文件集这意味着名为Samplestub.java一个文件不会被排除在文件集

## 模式集

一个模式集是一个模式，可以非常方便地筛选基于某种模式的文件或文件夹。可以使用下面的元字符来创建模式。

- ? - 只匹配一个字符
- 
  - 
    - 匹配零个或多个字符
- ** - 匹配零个或多个目录递归

下面的例子应该给一个模式集的用法的想法。

```
<patternset id="java.files.without.stubs">
  <include name="src/**/*.java"/>
  <exclude name="src/**/*Stub*"/>
</patternset>
```

patternset 可以用一个文件集重用如下：

```
<fileset dir="${src}" casesensitive="yes">
  <patternset refid="java.files.without.stubs"/>
 </fileset>
```

# 文件列表

在文件列表的数据类型类似设置，除了在文件列表中包含显式命名的文件列表，不支持通配符的文件

文件列表和文件组的数据类型之间的另一个主要区别是，在文件列表的数据类型可应用于可能会或可能还不存在的文件。

以下是文件列表的数据类型的一个例子

```
<filelist id="config.files" dir="${webapp.src.folder}">
  <file name="applicationConfig.xml"/>
  <file name="faces-config.xml"/>
  <file name="web.xml"/>
  <file name="portlet.xml"/>
</filelist>
```

在上面的例子中webapp.src.folder属性指向该项目的Web应用程序的源文件夹。

# 过滤器集

使用与复制任务筛选器集的数据类型，你可以匹配一个替代值的模式，所有的文件替换一定的文本。

一个常见的例子是附加版本号的发行说明文件，如下面的示例所示

```
<copy todir="${output.dir}">
  <fileset dir="${releasenotes.dir}" includes="**/*.txt"/>
  <filterset>
    <filter token="VERSION" value="${current.version}"/>
  </filterset>
</copy>
```

在上面的例子中output.dir属性指向项目的输出文件夹。

在上面的例子点releasenotes.dir属性的发行说明的项目文件夹中。

在上面的例子中current.version属性指向的项目的当前版本中的文件夹。

副本任务，顾名思义是用来从一个位置复制到另一个文件。

# 路径

path 数据类型通常用来代表一个类路径。在路径项用分号或冒号隔开。然而，这些字符会被正在运行的系统的路径分隔符替换一个运行时间。

最常见的类路径设置为项目中的jar文件和类的列表，如下面的例子：

```
<path id="build.classpath.jar">
  <pathelement path="${env.J2EE_HOME}/${j2ee.jar}"/>
  <fileset dir="lib">
      <include name="**/*.jar"/>
  </fileset>
</path>
```

在上面的例子中env.J2EE_HOME属性指向环境变量J2EE_HOME。

在上面的例子中的j2ee.jar属性指向在J2EE基础文件夹J2EE的jar文件的名称。

# Ant构建项目 - ANT

现在，我们已经了解了Ant数据类型，现在是时候把这些转化为行动。考虑下面的项目结构

项目将形成的Hello World传真应用程序项目在本教程的其余部分。

```
C:workFaxWebApplication>tree
Folder PATH listing
Volume serial number is 00740061 EC1C:ADB1
C:.
+---db
+---src
.   +---faxapp
.       +---dao
.       +---entity
.       +---util
.       +---web
+---war
    +---images
    +---js
    +---META-INF
    +---styles
    +---WEB-INF
        +---classes
        +---jsp
        +---lib
```

解释一下项目结构。

- 数据库脚本存储在 db 文件夹。
- Java源代码存储在src文件夹。
- 图像，JS，META-INF，样式（CSS）被存储在 war 文件夹。
- JSP被保存在jsp中文件夹。
- 第三方jar文件都存储在lib文件夹。
- Java类文件将被存储在WEB-INF classes文件夹。

这个练习的目的是建立一个编译的java类，并将它们放置在WEB-INF classes文件夹Ant文件。

下面是项目所需的build.xml文件。让我们看看它内容：

```
<?xml version="1.0"?>
<project name="fax" basedir="." default="build">
    <property name="src.dir" value="src"/>
    <property name="web.dir" value="war"/>
    <property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
    <property name="name" value="fax"/>

    <path id="master-classpath">
        <fileset dir="${web.dir}/WEB-INF/lib">
            <include name="*.jar"/>
        </fileset>
        <pathelement path="${build.dir}"/>
    </path>

    <target name="build" description="Compile source tree java file
        <mkdir dir="${build.dir}"/>
        <javac destdir="${build.dir}" source="1.5" target="1.5">
            <src path="${src.dir}"/>
            <classpath refid="master-classpath"/>
        </javac>
    </target>

    <target name="clean" description="Clean output directories">
        <delete>
            <fileset dir="${build.dir}">
                <include name="**/*.class"/>
            </fileset>
        </delete>
    </target>
</project>
```

首先，让我们为源，网上声明某些属性，并建立文件夹。

```
<property name="src.dir" value="src"/>
<property name="web.dir" value="war"/>
<property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
```

在这个例子中，对于src.dir是指项目（即，这里的java源文件可以找到）的源文件夹。

web.dir 指的是项目的网页源文件夹。在这里，您可以找到JSP，web.xml，CSS，JavaScript和其他Web相关的文件

最后，build.dir是指在项目编译的输出文件夹。

属性可以参考其他属性。如图所示，在上述例子中，build.dir属性使得参考web.dir属性。

在这个例子中，对于src.dir是指项目的源的文件夹。

我们的项目的默认目标compile目标。但首先，让我们看看clean目标。

clean目标，顾名思义删除build文件夹中的文件。

```
<target name="clean" description="Clean output directories">
   <delete>
      <fileset dir="${build.dir}">
         <include name="**/*.class"/>
      </fileset>
   </delete>
</target>
```

在主类路径保存在类路径的信息。在这种情况下，它包含在build文件夹中的类和在lib文件夹中的jar文件。

```
<path id="master-classpath">
   <fileset dir="${web.dir}/WEB-INF/lib">
      <include name="*.jar"/>
   </fileset>
   <pathelement path="${build.dir}"/>
</path>
```

最后，构建目标构建文件。首先，我们创建构建目录，如果它不存在。然后我们执行javac命令（指定JDK1.5作为我们的目标编译）。我们提供的源文件夹和类路径javac任务，并要求它砸在build文件夹中的类文件。

```
<target name="build" description="Compile main source tree java fi
   <mkdir dir="${build.dir}"/>
   <javac destdir="${build.dir}" source="1.5" target="1.5" debug="t
             deprecation="false" optimize="false" failonerror="true
      <src path="${src.dir}"/>
      <classpath refid="master-classpath"/>
   </javac>
</target>
```

在这个文件运行ant将编译java源文件，并将类build文件夹中。

下面的结果是运行Ant文件的结果：

```
C:>ant
Buildfile: C:uild.xml

BUILD SUCCESSFUL
Total time: 6.3 seconds
```

该文件被编译并放置在build.dir文件夹中。

# Ant构建文档 - ANT

文档是一个为任何项目所必须的。文档在项目的维护起到了极大的作用。通过使用内置的javadoc工具的Java使得文档更容易。Ant使得它甚至产生对需求文档更容易。

如你所知，javadoc工具具有高度的灵活性，并允许一些配置选项。Ant通过Javadoc任务公开这些配置选项。如果您不熟悉javadoc，建议在开始使用此Java文档教程。

以下部分列出了使用的Ant最常用的Javadoc选项。

## 属性

源可以使用源路径，sourcepathref或将源文件的规定。源路径是用来指向源文件（如src文件夹）的文件夹。 Sourcepathref用于参考，是由路径属性（例如，delegates.src.dir）引用的路径。而当你要指定的单个文件以逗号分隔的列表时的源文件使用。

使用destdir文件夹（例如build.dir）指定的目标路径

你可以由指定要包含在包名过滤Javadoc任务。这是通过使用packagenames属性来实现，用逗号分隔的包文件列表。

你可以过滤javadoc的过程中，只显示了公共，私有，包装或保护类和成员。这是通过使用（不奇怪）的私有，公共，封装和保护的属性来实现。

你也可以告诉javadoc的任务，包括使用相应属性的作者和版本信息。

你也可以组包一起使用的组属性，因此，它是易于浏览。

## 全部放在一起

让我们继续我们的主题 Hello world Fax 应用程序。让我们添加一个文件的目标我们的传真应用程序项目。

下面是我们的项目中使用的例子Javadoc任务。

```
<target name="generate-javadoc">
    <javadoc packagenames="faxapp.*" sourcepath="${src.dir}"
        destdir="doc" version="true" windowtitle="Fax Application">
      <doctitle><![CDATA[= Fax Application =]]></doctitle>
      <bottom>
         <![CDATA[Copyright © 2011\. All Rights Reserved.]]>
      </bottom>
      <group title="util packages" packages="faxapp.util.*"/>
      <group title="web packages" packages="faxapp.web.*"/>
      <group title="data packages"
                            packages="faxapp.entity.*:faxapp.dao.*"/>
    </javadoc>
    <echo message="java doc has been generated!" />
</target>
```

在这个例子中，我们指定使用对于src.dir作为源目录和文档作为目标目录中的javadoc。我们还定制了窗口标题，页眉和出现的Java文档页面上的页脚信息。

此外，我们已经创建了三组。一个用于为用户界面类和一组数据库相关类在我们的源文件夹，一组实用程序类。你可能会注意到数据包组有两个包 - faxapp.entity和faxapp.dao。

运行javadoc的Ant任务现在将生成并放置在doc文件夹中的Java文档文件。

当执行javadoc target，它会产生以下结果：

```
C:>ant generate-javadoc
Buildfile: C:uild.xml

java doc has been generated!

BUILD SUCCESSFUL
Total time: 10.63 second
```

Java文档文件现在出现在doc文件夹。

通常情况下，产生的释放或包的目标部分的Javadoc文件。

# Ant创建JAR文件 - ANT

编译Java源文件后的下一个合乎逻辑的步骤，是建立在Java归档，JAR文件。创建JAR文件与Ant用jar任务很容易。以下展示的是jar任务的常用属性

| 属性 | 描述 |
|---|---|
| basedir | The base directory for the output JAR file. By default, this is set to the base directory of the project. |
| compress | Advises ant to compress the file as it creates the JAR file. |
| keepcompression | While the **compress** attribute is applicable to the individual files, the**keepcompression** attribute does the same thing, but it applies to the entire archive. |
| destfile | The name of the output JAR file |
| duplicate | Advises Ant on what to do when duplicate files are found. You could add, preserve or fail the duplicate files. |
| excludes | Advises Ant to not include these comma separated list of files in the package. |
| excludesfile | Same as above, except the exclude files are specified using a pattern. |
| inlcudes | Inverse of excludes |
| includesfile | Inverse of excludesfile. |
| update | Advises ant to overwrite files in the already built JAR file. |

继续我们的Hello World传真应用程序项目，让我们添加一个新的目标，产生的jar文件。但在此之前，让我们考虑一下jar任务：

```
<jar destfile="${web.dir}/lib/util.jar"
     basedir="${build.dir}/classes"
     includes="faxapp/util/**"
     excludes="**/Test.class"
/>
```

在这个例子中，web.dir属性指向的网页源文件的路径。在我们的例子中，这是其中的util.jar将被放置。

在这个例子中，build.dir属性指向build文件夹在哪里可以找到 util.jar的类文件。

在这个例子中，我们创建了一个名为util.jar使用的类从faxapp.util一个jar文件。* 包。然而，我们不包括用名称测试结束课程。输出的jar文件会发生在webapp的lib 文件夹。

如果我们想使util.jar一个可执行JAR文件，我们需要添加清单与主Classmeta属性。

因此，上面的例子将被更新为：

```
<jar destfile="${web.dir}/lib/util.jar"
     basedir="${build.dir}/classes"
     includes="faxapp/util/**"
     excludes="**/Test.class">
     <manifest>
      <attribute name="Main-Class" value="com.yiibai.util.FaxUtil"/
     </manifest>
</jar>
```

要执行jar任务，一个目标里面把它包（最常见，构建或包的目标，并运行它们。

```
<target name="build-jar">
<jar destfile="${web.dir}/lib/util.jar"
     basedir="${build.dir}/classes"
     includes="faxapp/util/**"
     excludes="**/Test.class">
     <manifest>
      <attribute name="Main-Class" value="com.yiibai.util.FaxUtil"/
     </manifest>
</jar>
</target>
```

在这个文件运行Ant会为我们创建util.jar文件

下面的结果是运行Ant文件的结果：

```
C:>ant build-jar
Buildfile: C:uild.xml

BUILD SUCCESSFUL
Total time: 1.3 seconds
```

现在的util.jar文件放置在输出文件夹。

# Ant创建WAR文件 - ANT

创建WAR文件与Ant是非常简单，非常类似于创建JAR文件的任务。毕竟WAR文件是像JAR文件只是另一个ZIP文件。

WAR任务是一个扩展的JAR任务，但它有一些很好的补充操纵什么进入的WEB-INF/classes文件夹中，并生成web.xml文件。在WAR的任务是非常有用的指定WAR文件的特定布局。

由于WAR的任务是jar任务的扩展，jar任务的所有属性应用到WAR任务。下面是被指定到WAR任务的扩展属性：

| Attributes | 描述 |
|---|---|
| webxml | Path to the web.xml file |
| lib | A grouping to specify what goes into the WEB-INFlib folder. |
| classes | A grouping to specify what goes into the WEB-INFclasses folder. |
| metainf | Specifies the instructions for generating the MANIFEST.MF file. |

继续我们Hello World传真应用程序项目，让我们添加一个新的目标，产生的jar文件。但在此之前，让我们考虑WAR任务。请看下面的例子：

```
<war destfile="fax.war" webxml="${web.dir}/web.xml">
   <fileset dir="${web.dir}/WebContent">
      <include name="**/*.*"/>
   </fileset>
   <lib dir="thirdpartyjars">
      <exclude name="portlet.jar"/>
   </lib>
   <classes dir="${build.dir}/web"/>
</war>
```

按照前面的例子中，web.dir变量指的是源Web文件夹，即包含了JSP的文件夹，CSS，JavaScript文件等。

build.dir变量是指输出文件夹 - 这是在哪里可以找到类的WAR包。通常情况下，这些类将被捆绑到WAR文件的WEB-INF/classes文件夹中。

在这个例子中，我们创建一个名为fax.war 的 war文件。从网页源文件夹中获得的web.xml文件。所有从网上下了“WebContent”文件夹中的文件复制到WAR文件。

WEB-INF/lib文件夹中填充了来自第三方的jar文件夹中的jar文件。然而，我们不包括portlet.jar，因为这是已经存在于应用服务器的lib文件夹。最后，我们从构建目录的Web文件夹中复制所有的类和放入放在WEB-INF/classes文件夹中。

包裹WAR任务的Ant目标（通常包）内，并运行它。这将在指定的位置创建WAR文件。

这是完全可以嵌套类，库，metainf和WEBINF董事，使他们生活在分散的文件夹中的项目结构的任何地方。但最佳实践建议，您的Web项目应具备的Web内容结构类似于WAR文件的结构。Fax应用程序项目都有使用这个基本原则概述结构。

要执行WAR任务，一个目标里面把它包裹（最常见，构建或包的目标，并运行它们。

```
<target name="build-war">
   <war destfile="fax.war" webxml="${web.dir}/web.xml">
   <fileset dir="${web.dir}/WebContent">
      <include name="**/*.*"/>
   </fileset>
   <lib dir="thirdpartyjars">
      <exclude name="portlet.jar"/>
   </lib>
   <classes dir="${build.dir}/web"/>
</war>
</target>
```

在这个文件运行Ant会为我们创建了fax.war文件

下面的结果是运行Ant文件的结果：

```
C:>ant build-war
Buildfile: C:uild.xml

BUILD SUCCESSFUL
Total time: 12.3 seconds
```

现在的fax.war文件放置在输出文件夹。 war文件的内容将是：

```
fax.war:
  +---jsp             _This folder contains the jsp files_
  +---css             _This folder contains the stylesheet files_
  +---js              _This folder contains the javascript files_
  +---images          _This folder contains the image files_
  +---META-INF        _This folder contains the Manifest.Mf_
  +---WEB-INF
        +---classes   _This folder contains the compiled classes_
        +---lib       _Third party libraries and the utility jar fi
        WEB.xml       _Configuration file that defines the WAR pack
```

# Ant打包应用 - ANT

我们已经知道Ant使用的Hello World Fax 的Web应用程序中零碎的不同。

现在是时候把一切融合在一起，以创建一个完整的和完整的build.xml文件。考虑build.properties和build.xml文件列表如下：

## build.properties

```
deploy.path=c:    omcat6webapps
```

## build.xml

```
<?xml version="1.0"?>

<project name="fax" basedir="." default="usage">
    <property file="build.properties"/>
    <property name="src.dir" value="src"/>
    <property name="web.dir" value="war"/>
    <property name="javadoc.dir" value="doc"/>
    <property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
    <property name="name" value="fax"/>

    <path id="master-classpath">
        <fileset dir="${web.dir}/WEB-INF/lib">
            <include name="*.jar"/>
        </fileset>
        <pathelement path="${build.dir}"/>
    </path>

    <target name="javadoc">
      <javadoc packagenames="faxapp.*" sourcepath="${src.dir}"
      destdir="doc" version="true" windowtitle="Fax Application">
        <doctitle><![CDATA[<h1>= Fax Application
          =</h1>]]></doctitle>
        <bottom><![CDATA[Copyright © 2011\. All
          Rights Reserved.]]></bottom>
        <group title="util packages" packages="faxapp.util.*"/>
        <group title="web packages" packages="faxapp.web.*"/>
        <group title="data packages"
          packages="faxapp.entity.*:faxapp.dao.*"/>
      </javadoc>
    </target>

    <target name="usage">
```

```
        <echo message=""/>
        <echo message="${name} build file"/>
        <echo message="-----------------------------------"/>
        <echo message=""/>
        <echo message="Available targets are:"/>
        <echo message=""/>
        <echo message="deploy    --> Deploy application
          as directory"/>
        <echo message="deploywar --> Deploy application
          as a WAR file"/>
        <echo message=""/>
    </target>

    <target name="build" description="Compile main
      source tree java files">
        <mkdir dir="${build.dir}"/>
        <javac destdir="${build.dir}" source="1.5"
          target="1.5" debug="true"
          deprecation="false" optimize="false" failonerror="true">
            <src path="${src.dir}"/>
            <classpath refid="master-classpath"/>
        </javac>
    </target>

    <target name="deploy" depends="build"
      description="Deploy application">
        <copy todir="${deploy.path}/${name}"
          preservelastmodified="true">
            <fileset dir="${web.dir}">
                <include name="**/*.*"/>
            </fileset>
        </copy>
    </target>

    <target name="deploywar" depends="build"
      description="Deploy application as a WAR file">
        <war destfile="${name}.war"
             webxml="${web.dir}/WEB-INF/web.xml">
            <fileset dir="${web.dir}">
                <include name="**/*.*"/>
            </fileset>
        </war>
        <copy todir="${deploy.path}" preservelastmodified="true">
            <fileset dir=".">
                <include name="*.war"/>
            </fileset>
        </copy>
    </target>

    <target name="clean" description="Clean output directories">
        <delete>
            <fileset dir="${build.dir}">
                <include name="**/*.class"/>
```

```
            </fileset>
        </delete>
    </target>

</project>
```

在这个例子中，我们首先声明的路径的webapps文件夹中的Tomcat在生成属性文件作为deploy.path变量。我们还声明的源文件夹src.dir的变量的java文件。然后我们声明的源文件夹中web.dir变量的网页文件。 javadoc.dir是用于存储Java文档的文件夹，build.dir是用于存储生成的输出文件的路径。

然后我们声明的Web应用程序，这是 Fax 在我们的例子中的名称。

我们还定义了主类路径的conains存在于项目的WEB-INF/lib文件夹中的JAR文件。我们还包括了类文件呈现在build.dir在主类路径

Javadoc的目标生产项目所需的Javadoc和使用对象是用来打印，是目前在构建文件的共同目标。

上面的例子显示了两个部署目标 - 部署和deploywar目标

部署目标文件从web目录下的文件复制到deploy目录保留最后修改的日期时间戳记。部署到支持热部署一台服务器时，这是很有用的。

clean目标清除所有先前建立的文件。

部署war目标构建war文件，然后复制war文件到应用服务器的部署目录。

# Ant部署应用程序 - ANT

在前面的章节中，我们已经学会了如何打包应用程序并将其部署到一个文件夹中。在这个例子中，我们打算把它更进一步。

我们要部署Web应用程序直接到应用程序的服务器的部署文件夹，然后我们将添加一些Ant目标来启动和停止服务。让我们继续的Hello World传真的Web应用程序。这是一个延续前一章，新元件在突出红色

## build.properties

```
# Ant properties for building the springapp

appserver.home=c:\install\apache-tomcat-7.0.19
# for Tomcat 5 use $appserver.home}/server/lib
# for Tomcat 6 use $appserver.home}/lib
appserver.lib=${appserver.home}/lib

deploy.path=${appserver.home}/webapps

tomcat.manager.url=http://www.yiibai.com:8080/manager
tomcat.manager.username=yiibai
tomcat.manager.password=secret
```

## build.xml

```
<?xml version="1.0"?>

<project name="fax" basedir="." default="usage">
    <property file="build.properties"/>
    <property name="src.dir" value="src"/>
    <property name="web.dir" value="war"/>
    <property name="javadoc.dir" value="doc"/>
    <property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
    <property name="name" value="fax"/>

    <path id="master-classpath">
        <fileset dir="${web.dir}/WEB-INF/lib">
            <include name="*.jar"/>
        </fileset>
        <pathelement path="${build.dir}"/>
    </path>

    <target name="javadoc">
```

```
        <javadoc packagenames="faxapp.*" sourcepath="${src.dir}"
        destdir="doc" version="true" windowtitle="Fax Application">
          <doctitle><![CDATA[<h1>= Fax Application
            =</h1>]]></doctitle>
          <bottom><![CDATA[Copyright © 2011\. All
            Rights Reserved.]]></bottom>
          <group title="util packages" packages="faxapp.util.*"/>
          <group title="web packages" packages="faxapp.web.*"/>
          <group title="data packages"
            packages="faxapp.entity.*:faxapp.dao.*"/>
        </javadoc>
      </target>

      <target name="usage">
          <echo message=""/>
          <echo message="${name} build file"/>
          <echo message="-----------------------------------"/>
          <echo message=""/>
          <echo message="Available targets are:"/>
          <echo message=""/>
          <echo message="deploy    --> Deploy application
            as directory"/>
          <echo message="deploywar --> Deploy application
            as a WAR file"/>
          <echo message=""/>
      </target>

      <target name="build" description="Compile main
        source tree java files">
          <mkdir dir="${build.dir}"/>
          <javac destdir="${build.dir}" source="1.5"
            target="1.5" debug="true"
            deprecation="false" optimize="false" failonerror="true">
              <src path="${src.dir}"/>
              <classpath refid="master-classpath"/>
          </javac>
      </target>

      <target name="deploy" depends="build"
        description="Deploy application">
          <copy todir="${deploy.path}/${name}"
            preservelastmodified="true">
              <fileset dir="${web.dir}">
                  <include name="**/*.*"/>
              </fileset>
          </copy>
      </target>

      <target name="deploywar" depends="build"
        description="Deploy application as a WAR file">
          <war destfile="${name}.war"
               webxml="${web.dir}/WEB-INF/web.xml">
              <fileset dir="${web.dir}">
```

```
                <include name="**/*.*"/>
            </fileset>
        </war>
        <copy todir="${deploy.path}" preservelastmodified="true">
            <fileset dir=".">
                <include name="*.war"/>
            </fileset>
        </copy>
    </target>

    <target name="clean" description="Clean output directories">
        <delete>
            <fileset dir="${build.dir}">
                <include name="**/*.class"/>
            </fileset>
        </delete>
    </target>
```

```
    <!-- ============================================================
    <!-- Tomcat tasks -->
    <!-- ============================================================

    <path id="catalina-ant-classpath">
        <!-- We need the Catalina jars for Tomcat -->
        <!--  * for other app servers - check the docs -->
        <fileset dir="${appserver.lib}">
            <include name="catalina-ant.jar"/>
        </fileset>
    </path>

    <taskdef name="install"
        classname="org.apache.catalina.ant.InstallTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>
    <taskdef name="reload"
        classname="org.apache.catalina.ant.ReloadTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>
    <taskdef name="list"
        classname="org.apache.catalina.ant.ListTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>
    <taskdef name="start"
        classname="org.apache.catalina.ant.StartTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>
    <taskdef name="stop"
         classname="org.apache.catalina.ant.StopTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>

    <target name="reload" description="Reload application in Tomcat
        <reload url="${tomcat.manager.url}"
                 username="${tomcat.manager.username}"
                 password="${tomcat.manager.password}"
                 path="/${name}"/>
    </target>
</project>
```

在本练习中，我们使用Tomcat作为我们的应用服务器。首先，在构建属性文件中，我们定义了一些附加属性。

- appserver.home应用指向安装路径到Tomcat应用服务器。
- appserver.lib指向Tomcat的安装文件夹中的库文件。
- deploy.path变量现在指向Tomcat中的web应用文件夹。

在Tomcat中的应用程序可以停止和startedusing Tomcat管理应用程序。也是在build.properties文件中指定的URL管理器应用程序，使用用户名和密码。接下来我们声明一个新的CLASSPATH中包含catalina-ant.jar文件。这个jar文件是必需通过Apache Ant来执行Tomcat 任务。

catalina-ant.jar 文件提供了以下任务：

| Properties | 描述 |
|---|---|
| InstallTask | Installs a web application. Class Name: org.apache.catalina.ant.InstallTask |
| ReloadTask | Reload a web application. Class Name: org.apache.catalina.ant.ReloadTask |
| ListTask | Lists all web applications. Class Name: org.apache.catalina.ant.ListTask |
| StartTask | Starts a web application. Class Name: org.apache.catalina.ant.StartTask |
| StopTask | Stops a web application. Class Name: org.apache.catalina.ant.StopTask |
| ReloadTask | Reloads a web application without stopping. Class Name: org.apache.catalina.ant.ReloadTask |

重装任务需要以下附加参数。

- 1）URL到管理器应用程序 2）用户名重新启动Web应用程序 3）密码重新启动Web应用程序重新启动Web应用程序 4）名称

让我们发出的deploy-war命令的web应用程序复制到Tomcat的webapps文件夹中，然后让我们重新加载该传真的Web应用程序。下面的结果是运行Ant文件的结果：

```
C:>ant deploy-war
Buildfile: C:uild.xml

BUILD SUCCESSFUL
Total time: 6.3 seconds

C:>ant reload
Buildfile: C:uild.xml

BUILD SUCCESSFUL
Total time: 3.1 seconds
```

一旦上述任务运行时，Web应用程序部署和Web应用程序重新加载。

# Ant执行Java代码 - ANT

您可以使用Ant来执行java代码。在下面这个例子中，java类中取一个参数（管理员的电子邮件地址），并发送了一封电子邮件。

```
public class NotifyAdministrator
{
  public static void main(String[] args)
  {
    String email = args[0];
    notifyAdministratorviaEmail(email);
    System.out.println("Administrator "+email+" has been notified")
  }
  public static void notifyAdministratorviaEmail(String email)
  {
      //......
  }
}
```

下面是执行这个java类简单的构建。

```
<?xml version="1.0"?>
<project name="sample" basedir="." default="notify">
  <target name="notify">
    <java fork="true" failonerror="yes" classname="NotifyAdministra
      <arg line="admin@test.com"/>
    </java>
  </target>
</project>
```

当执行构建时，它会产生以下结果：

```
C:>ant
Buildfile: C:uild.xml

notify:
     [java] Administrator admin@test.com has been notified

BUILD SUCCESSFUL
Total time: 1 second
```

在这个例子中，java代码做一个简单的事情 - 发送电子邮件。我们也可以使用内置的Ant任务来做到这一点。不过，现在你已经得到了你的想法可以扩展你的构建文件来调用java代码执行复杂的东西，例如：加密你的源代码。

# Ant和Eclipse集成 - ANT

如果您已经下载并已经安装了Eclipse，你很少做上手。 Eclipse中预装捆绑在一起的Ant的插件，随时供您使用。

按照简单的步骤，到Ant集成到Eclipse中。

- 确保build.xml文件就是java项目的一部分，并没有生活在一个位置，是外部的项目。
- 通过将启用Ant视图 Window > Show View > Other > Ant > Ant
- 打开项目资源管理器中，拖动的build.xml到Ant视图

Ant视图中显示现在看起来类似于：

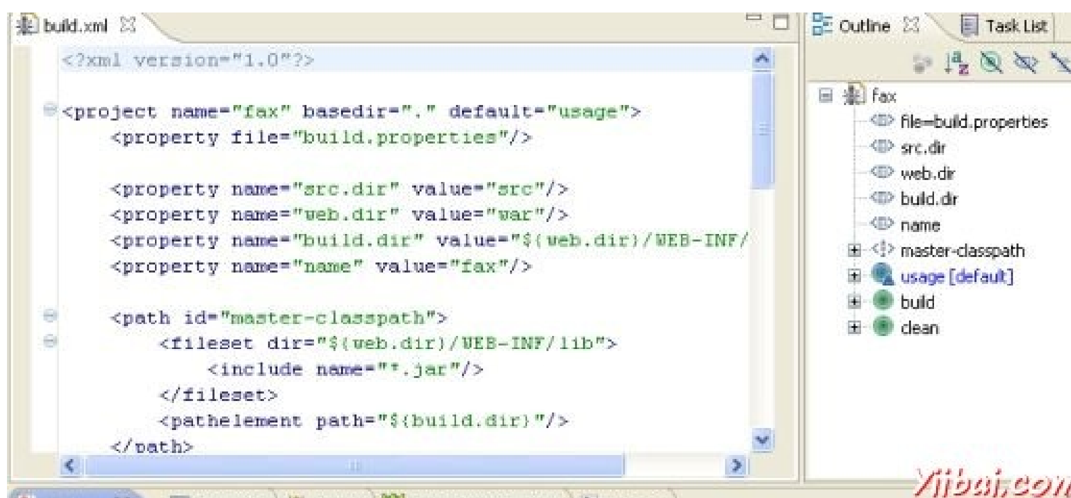



点击目标，build / clean / usage将运行Ant你的目标。

点击"fax"， 将执行默认的目标 - usage

Ant的Eclipse插件还附带了一个很好的编辑器来编辑的build.xml文件。该编辑器是知道的build.xml架构，可以帮助您提供代码完成。

要使用Ant编辑器，右键单击您的build.xml（从项目资源管理器），然后选择打开方式>Ant Editor。 Ant Editor应该loook类似于：



Ant 编辑器列出了右手边的目标，该目标列表作为一个书签，让你直接跳到编辑特定的目标。

# Ant Junit集成 - ANT

JUnit 是基于Java常用的单元测试框架进行开发。它是易于使用和易于延伸。有许多JUnit扩展可用。如果你不熟悉Junit的，你应该从www.junit.org下载JUnit和阅读JUnit的使用手册。

本教程讨论了关于执行使用Ant 的JUnit测试。Ant 通过这个简单Junit 的任务变得简单。

以下展示的是JUnit 任务的属性。

| Properties | 描述 |
| --- | --- |
| dir | Where to invoke the VM from. This is ignored when **fork** is disabled. |
| jvm | Command used to invoke the JVM. This is ignored when **fork** is disabled. |
| fork | Runs the test in a separate JVM |
| errorproperty | The name of the property to set if there is a Junit error |
| failureproperty | The name of the property to set if there is a Junit failure |
| haltonerror | Stops execution when a test error occurs |
| haltonfailure | Stops execution when a failure occurs |
| printsummary | Advices Ant to display simple statistics for each test |
| showoutput | Adivces Ant tosend the output to its logs and formatters |
| tempdir | Path to the temporary file that Ant will use |
| timeout | Exits the tests that take longer to run than this setting (in milliseconds). |

让我们继续的Hello World fax web应用程序的主题，并添加一个JUnit目标。

下面的例子展示了一个简单的JUnit测试执行

```
<target name="unittest">
  <junit haltonfailure="true" printsummary="true">
    <test name="com.yiibai.UtilsTest"/>
  </junit>
</target>
```

上面的例子显示的Junit对com.yiibai.UtilsTest JUnit类执行。运行上面会产生下面的输出

```
test:
[echo] Testing the application
[junit] Running com.yiibai.UtilsTest
[junit] Tests run: 12, Failures: 0, Errors: 0, Time elapsed: 16.2 s
BUILD PASSED
```

# POI教程

很多时候，一个软件应用程序需要生成Microsoft Excel文件格式的报告。有时，一个应用程序甚至希望将Excel文件作为输入数据。例如，一个公司开发的应用程序将财务部门需要所有输出生成自己的Excel。

任何Java程序员愿意将MS Office文件的输出，可以使用预定义和只读API来做到。

## 什么是Apache POI？

Apache POI是一种流行的API，它允许程序员使用Java程序创建，修改和显示MS Office文件。这由Apache软件基金会开发使用Java分布式设计或修改Microsoft Office文件的开源库。它包含类和方法对用户输入数据或文件到MS Office文档进行解码。

## Apache POI组件

Apache POI包含类和方法，来将MS Office所有OLE 2文档复合。此API组件的列表如下。

- **POIFS (较差混淆技术实现文件系统) :** 此组件是所有其他POI元件的基本因素。它被用来明确地读取不同的文件。
- **HSSF (可怕的电子表格格式) :** 它被用来读取和写入MS-Excel文件的xls格式。
- **XSSF (XML格式) :** 它是用于MS-Excel中XLSX文件格式。
- **HPSF (可怕的属性设置格式) :** 它用来提取MS-Office文件属性设置。
- **HWPF (可怕的字处理器格式) :** 它是用来读取和写入MS-Word的文档扩展名的文件。
- **XWPF (XML字处理器格式) :** 它是用来读取和写入MS-Word的docx扩展名的文件。
- **HSLF (可怕的幻灯片版式格式) :** 它是用于读取，创建和编辑PowerPoint演示文稿。
- **HDGF (可怕的图表格式) :** 它包含类和方法为MS-Visio的二进制文件。
- **HPBF (可怕的出版商格式) :** 它被用来读取和写入MS-Publisher文件。

本教程将指导使用Java Excel文件完成工作过程。因此，本教程仅限于HSSF和XSSF组件。

注：旧版本的POI支持二进制文件格式，如DOC，XLS，PPT等从版本3.5起，POI支持微软Office的OOXML文件格式，如DOCX，XLSX，PPTX等。

如Apache POI，还有由不同的供应商为Excel文件的生成提供的其他库。这些措施包括Aspose面向Java的Aspose，JXL 通过共享库由JExcel团队开发。

# Apache POI - Java Excel APIs - POI教程

本章将介绍一些Java ExcelAPI和它们的特征。有许多厂商谁提供Java Excel相关的API;其中一些将在这一章中讨论。

## Java Aspose Cells

Java Aspose Cells 是一种纯粹的Java授权的Excel API，开发和供应商Aspose发布。这个API的最新版本是8.1.2，发布于2014年7月，是一个丰富而厚重的API(普通Java类和AWT类的组合)设计，可以读、写和操纵电子表格Excel的组件。此API常见用途如下：

- Excel报表，建立动态Excel报表
- 高保真的Excel渲染和打印
- 从Excel电子表格中导入和导出数据
- 生成，编辑，转换和电子表格

## JXL

JXL是一款专为Selenium第三方框架，支持基于Web浏览器(数据是Web浏览器自动更新)数据驱动的自动化。然而，它也被用来作为JExcel API的一个共同的支持库，因为它的基本功能是可创建，读取和写入电子表格。基本特征如下：

- 生成Excel文件
- 从工作簿和电子表格导入数据
- 获得行和列的总数

注意：JXL只支持xls档案格式，并且不能处理大数据量。

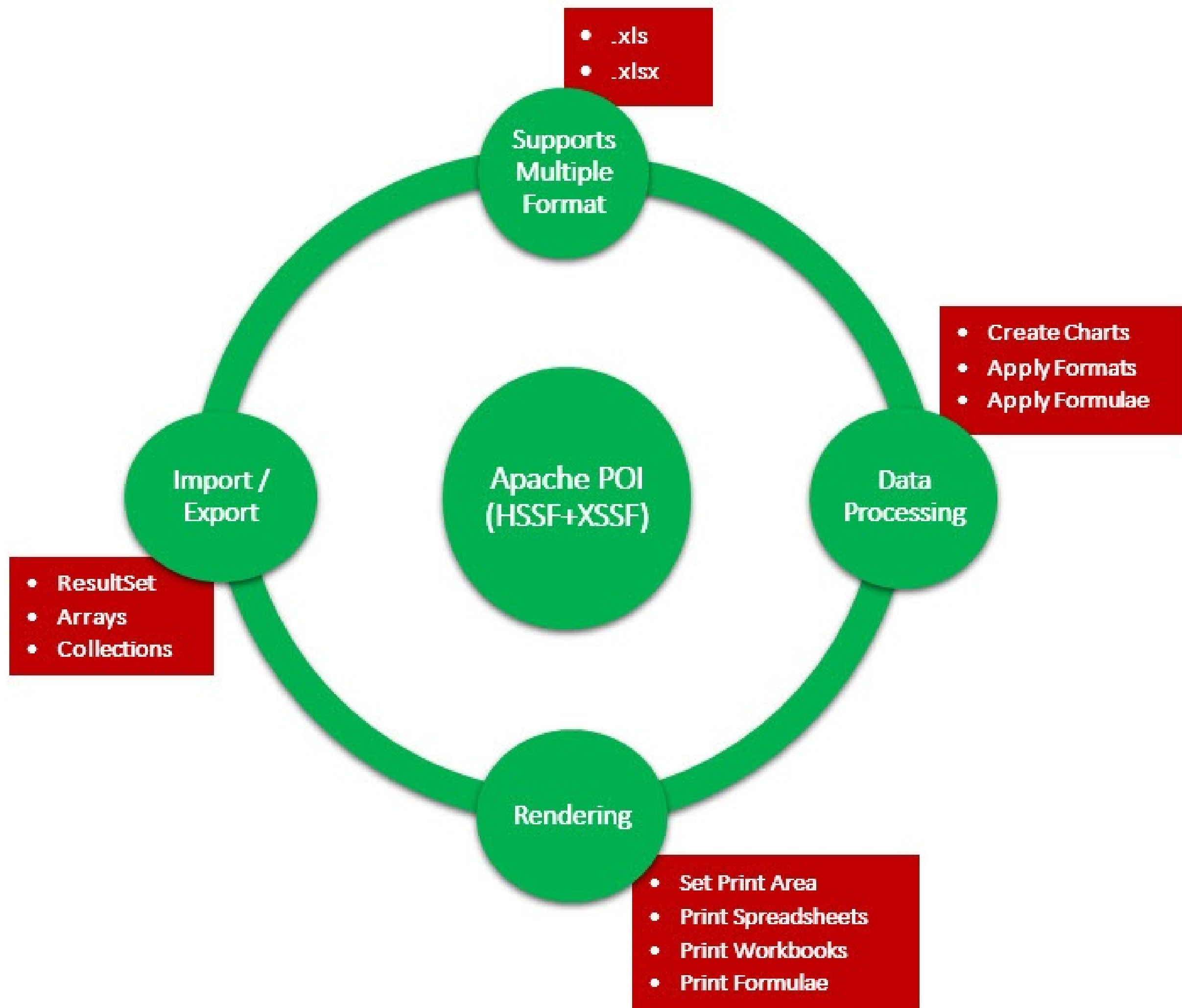
## JExcel

JExcel是由Team Dev开发提供纯行货API。利用这一点程序员可以很容易地读取，写入，显示和修改Excel工作簿中的两种格式：.xls和.XLSX。这个API可以很容易地嵌入Java的Swing和AWT。这个API的最新版本是Jexcel-2.6.12，发布于2009年，主要特点如下。

- 自动化Excel应用程序，工作簿，工作表等
- 在Java Swing应用程序作为普通的Swing组件嵌入到工作簿
- 事件侦听器添加到工作簿和电子表格
- 添加事件处理程序来处理的工作簿和电子表格事件的行为
- 添加本地对等开发定制功能

# Apache POI

Apache POI是Apache软件基金会提供的100%开源库。大多数中小规模的应用程序开发主要依赖于Apache POI（HSSF+ XSSF）。它支持Excel 库的所有基本功能; 然而，呈现和文本提取是它的主要特点。



# Apache POI环境设置 - POI教程

本章将指导完成Apache POI在Windows和Linux系统为基础的设置过程。 Apache POI可以轻松地安装和集成,下面没有任何复杂的设置过程,通过几个简单步骤，目前Java环境，用户管理是必需安装的。

## 系统要求

| JDK | Java SE 2 JDK 1.5 或以上 |
|---|---|
| 内存 | 1 GB RAM (推荐) |
| 磁盘空间 | 没有最小要求 |
| 操作系统版本 | Windows XP 或以上, Linux |

现在让我们继续安装Apache POI 的步骤。

## 第1步：验证Java安装

首先，需要在系统上安装Java软件开发工具包(SDK)。为了验证这一点，执行任何根据使用的平台上的以下两个命令。

如果Java安装已完成正确，那么它会显示Java安装的当前版本和规范。样本输出给下表中。

| 平台 | 命令 | 样本输出 |
|---|---|---|
| Windows | 打开命令控制台然后键入: `\>java -version` | Java version "1.7.0_60" Java (TM) SE Run Time Environment (build 1.7.0_60-b19) Java Hotspot (TM) 64-bit Server VM (build 24.60-b09,mixed mode) |
| Linux | 打开命令终端，输入：$java –version | java version "1.7.0_25" Open JDK Runtime Environment (rhel-2.3.10.4.el6_4-x86_64) Open JDK 64-Bit Server VM (build 23.7-b01, mixed mode) |

- 我们假设本教程的读者安装的是Java SDK版本1.7.0_60安装在他们的系统中。
- 如果没有Java SDK，从 http://www.oracle.com/technetwork/java/javase/downloads/index.html 下载其当前版本并安装它。

## 第2步：设置Java环境

设置环境变量JAVA_HOME指向安装了机器上Java的基本目录位置。例如，

| 平台 | 描述 |
| --- | --- |
| Windows | Set JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60 |
| Linux | Export JAVA_HOME=/usr/local/java-current |

添加Java编译器位置的完整路径到系统路径。

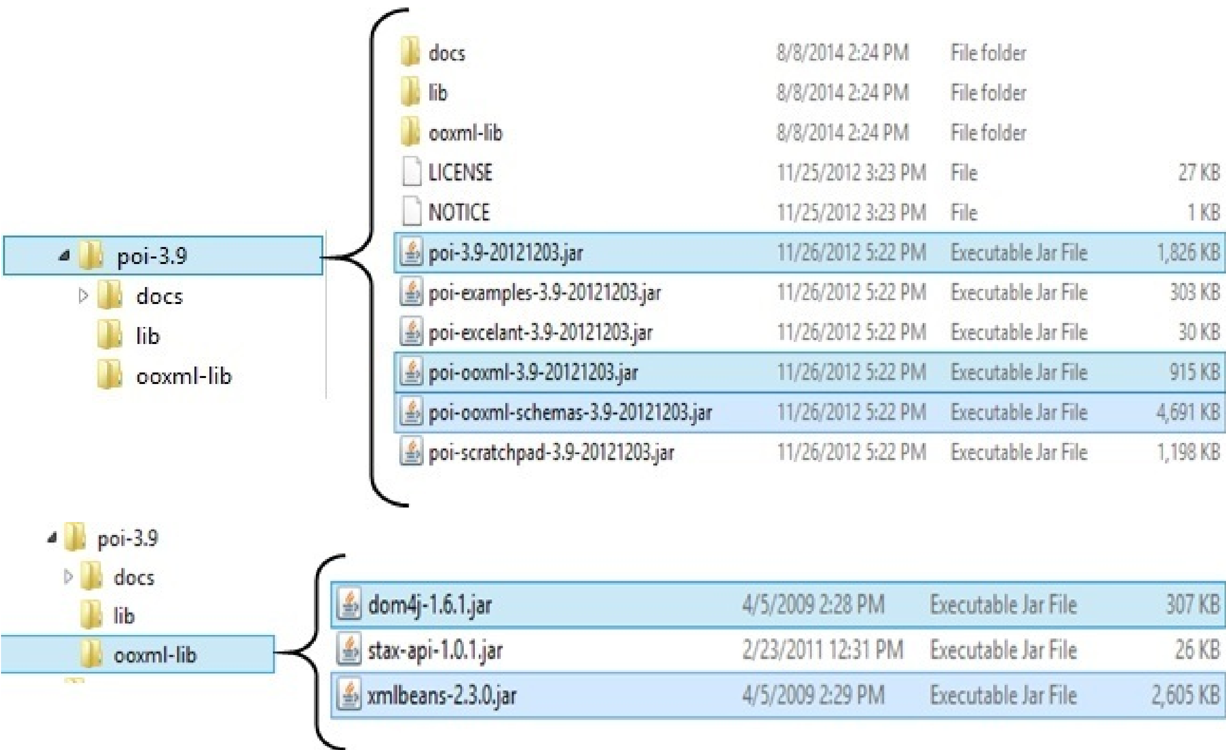
| 平台 | 描述 |
| --- | --- |
| Windows | Linux |
| 添加字符串 "C:\Program Files\Java\jdk1.7.0_60\bin" 到系统环境变量PATH 的尾部。 | Export PATH=$PATH:$JAVA_HOME/bin/ |

从命令提示符下执行命令java-version如上所述。

## 第3步：安装Apache POI库

从 http://poi.apache.org/download.htmll 下载Apache POI的最新版本，并解压缩所需要的库，可以链接到Java程序的文件夹。我们假设该文件在C盘的文件夹中。

下面的图像显示所下载的文件夹内的目录和文件结构。



添加的五个jar文件的完整路径，在上面的图片路径到CLASSPATH。

Windows: 添加以下字符串到用户变量的末尾

```
CLASSPATH:
“C:\poi-3.9\poi-3.9-20121203.jar;”
“C:\poi-3.9\poi-ooxml-3.9-20121203.jar;”
“C:\poi-3.9\poi-ooxml-schemas-3.9-20121203.jar;”
“C:\poi-3.9\ooxml-lib\dom4j-1.6.1.jar;”
“C:\poi-3.9\ooxml-lib\xmlbeans-2.3.0.jar;.;”
```

Linux:

```
Export CLASSPATH=$CLASSPATH:
/usr/share/poi-3.9/poi-3.9-20121203.tar:
/usr/share/poi-3.9/poi-ooxml-schemas-3.9-20121203.tar:
/usr/share/poi-3.9/poi-ooxml-3.9-20121203.tar:
/usr/share/poi-3.9/ooxml-lib/dom4j-1.6.1.tar:
/usr/share/poi-3.9/ooxml-lib/xmlbeans-2.3.0.tar
```

到这里，整个POI环境设置已经完成，下一节我们来学习如何使用。

# POI核心类 - POI教程

本章介绍了Apache POI的API，它是至关重要的工作，使用Java程序操作Excel文件有下面几个类和方法。

## 工作簿

这是创建或维护Excel工作簿的所有类的超接口。它属于org.apache.poi.ss.usermodel包。是实现此接口的两个类，如下所示：

- HSSFWorkbook：这个类有读取和.xls 格式和写入Microsoft Excel文件的方法。它与微软Office97-2003版本兼容。
- XSSFWorkbook：这个类有读写Microsoft Excel和OpenOffice的XML文件的格式.xls或.xlsx的方法。它与MS-Office版本2007或更高版本兼容。

## HSSFWorkbook

它是在org.apache.poi.hssf.usermodel包的高层次的类。它实现了Workbook 接口，用于Excel文件中的.xls格式。下面列出的是一些本类下的方法和构造函数。

### 类的构造函数

| S.No. | 构造函数和说明 |
|---|---|
| 1 | **HSSFWorkbook()** 从头开始创建一个新的HSSFWorkbook对象时。 |
| 2 | **HSSFWorkbook(DirectoryNode directory, boolean preserveNodes)** 创建一个特定的目录中一个新的HSSFWworkbook对象。 |
| 3 | **HSSFWorkbook(DirectoryNode directory, POIFSFileSystem fs, boolean preserveNodes)** 给定一个POIFSFileSystem对象和特定的目录中，它创建了一个SSFWorkbook对象读取指定的工作簿。 |
| 4 | **HSSFWorkbook(java.io.InputStream s)** 创建使用输入流中的新HSSFWorkbook对象时。 |
| 5 | **HSSFWorkbook(java.io.InputStream s, boolean preserveNodes)** 构建在输入流的POI文件系统。 |
| 6 | **HSSFWorkbook(POIFSFileSystem fs)** 使用POIFSFileSystem对象构造的新HSSFWorkbook对象时。 |
| 7 | **HSSFWorkbook(POIFSFileSystem fs, boolean preserveNodes)** 给定一个POIFSFileSystem对象时，它会创建一个新的HSSFWorkbook对象时读取指定的工作簿。 |

这些构造内的常用参数：

- directory : 这是从POI文件系统处理的目录。
- fs :它是包含簿流该POI的文件系统。
- preservenodes : 这是决定是否保留其他节点像宏的可选参数。它消耗大量的内存，因为它存储在内存中的所有POIFileSystem(如果设置)。

注意：HSSFWorkbook类包含了许多方法;然而，它们仅与XLS格式兼容。在本教程中，重点是在Excel文件格式的最新版本。因此，HSSFWorkbook类的方法，这里没有列出。如果需要这些类的方法，那么请参照POI-HSSFWorkbook类API在https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFWorkbook.html.

# XSSFWorkbook

它是用于表示高和低层次Excel文件格式的类。它属于org.apache.xssf.usemodel包，并实现Workbook接口。下面列出的是这个类的方法和构造函数。

## 类的构造函数

| S.No. | 构造函数和说明 |
|---|---|
| 1 | **XSSFWorkbook()** 从头开始创建一个新的XSSFworkbook对象。 |
| 2 | **XSSFWorkbook(java.io.File file)** 构造从给定文件中的XSSFWorkbook对象。 |
| 3 | **XSSFWorkbook(java.io.InputStream is)** 构造一个XSSFWorkbook对象，通过缓冲整个输入流到内存中，然后为它打开一个OPCPackage对象。 |
| 4 | **XSSFWorkbook(java.lang.String path)** 构建一个给定文件的完整路径的XSSFWorkbook对象。 |

### 类方法

| S.No. | 方法及描述 |
|---|---|
| 1 | **createSheet()** 创建一个XSSFSheet本工作簿，将其添加到表，并返回高层表示。 |
| 2 | **createSheet(java.lang.String sheetname)** 创建此工作簿的新表，并返回高层表示。 |
| 3 | **createFont()** 创建一个新的字体，并将其添加到工作簿的字体表。 |
| 4 | **createCellStyle()** 创建一个新的XSSFCellStyle并将其添加到工作簿的样式表。 |
| 5 | **createFont()** 创建一个新的字体，并将其添加到工作簿的字体表。 |
| 6 | **setPrintArea(int sheetIndex, int startColumn, int endColumn, int startRow,int endRow)** 设置一个给定的表按照指定参数的打印区域。 |

对于此类的其余的方法，请参阅完整的API文档：http://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFWorkbook.html. 列出了所有方法。

## Sheet

Sheet是在org.apache.poi.ss.usermodel包的接口，它是创建具有特定名称的高或低级别的电子表格的所有类的超接口。电子表格的最常见的类型是工作表，它被表示为单元的网格。

## HSSFSheet

这是在org.apache.poi.hssf.usermodel包的类。它可以创建Excel电子表格，它允许在sheet 方式和表数据格式。

### 类的构造函数

| S.No. | 构造函数及描述 |
|---|---|
| 1 | **HSSFSheet(HSSFWorkbook workbook)** 创建新HSSFSheet通过调用HSSFWorkbook从头开始创建一个表。 |
| 2 | **HSSFSheet(HSSFWorkbook workbook, InternalSheet sheet)** 创建HSSFSheet表示给定表对象。 |

## XSSFSheet

这是代表了Excel电子表格的高级别代表的一类。这在org.apache.poi.hssf.usermodel包下。

### 类的构造函数

| S.No. | 构造函数及描述 |
|---|---|
| 1 | **XSSFSheet()** 创造了新的XSSFSheet- 调用XSSFWorkbook从头开始创建一个表。 |
| 2 | **XSSFSheet(PackagePart part, PackageRelationship rel)** 创建XSSFSheet表示给定包的一部分和关系。 |

### 类方法

| S.No. | 方法和描述 |
|---|---|
| 1 | **addMergedRegion(CellRangeAddress region)** 添加单元的合并区域（因此这些单元格合并形成一个）。 |
| 2 | **autoSizeColumn(int column)** 调整列宽，以适应的内容。 |
| 3 | **iterator()** 此方法是用于rowIterator()的别名，以允许foreach循环 |
| 4 | **addHyperlink(XSSFHyperlink hyperlink)** 注册超链接的集合中的超链接此工作表格上 |

对于此类的其余的方法，请参阅完整的API在：https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFSheet.html.

## 行

这是在org.apache.poi.ss.usermodel包的接口。它是用于一排的电子表格的高层表示。它是代表了POI库的行所有类的超接口。

# XSSFRow

这是在org.apache.poi.xssf.usermodel包的类。它实现了Row接口，因此它可以在电子表格中创建行。下面列出的是这个类在方法和构造函数。

## 类方法

| S.No. | 描述 |
| --- | --- |
| 1 | **createCell(int columnIndex)** 创建新单元行并返回。 |
| 2 | **setHeight(short height)** 设置短单位的高度。 |

对于此类的其余的方法，参考如下链接：
https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFRow.html

## 单元格

这是在org.apache.poi.ss.usermodel包的接口。它是代表了单元在电子表格中的行中的所有类的超接口。

单元格可以使用各种属性，例如空白，数字，日期，错误等单元格被添加到一个行之前应具有（基于0）自己的编号。

# XSSFCell

这是在 org.apache.poi.xssf.usermodel 包的类。它实现了单元格界面。它是单元在电子表格中的行的一个高层次的表示。

## 字段摘要

下面列出的是一些XSSFCell类的字段以及它们的描述。

| 单元格类型 | 描述 |
| --- | --- |
| CELL_TYPE_BLANK | 代表空白单元格 |
| CELL_TYPE_BOOLEAN | 代表布尔单元（true或false） |
| CELL_TYPE_ERROR | 表示在单元的误差值 |
| CELL_TYPE_FORMULA | 表示一个单元格公式的结果 |
| CELL_TYPE_NUMERIC | 表示对一个单元的数字数据 |
| CELL_TYPE_STRING | 表示对一个单元串（文本） |

### 类方法

| S.No. | 描述 |
|---|---|
| 1 | **setCellStyle(CellStyle style)** 为单元格设置样式。 |
| 2 | **setCellType(int cellType)** 设置单元格的类型（数字，公式或字符串）。 |
| 3 | **setCellValue(boolean value)** 设置单元格一个布尔值 |
| 4 | **setCellValue(java.util.Calendar value)** 设置一个日期值的单元格。 |
| 5 | **setCellValue(double value)** 设置为单元格的数值。 |
| 6 | **setCellValue(java.lang.String str)** 设置为单元格的字符串值。 |
| 7 | **setHyperlink(Hyperlink hyperlink)** 分配超链接到该单元格。 |

对于这个类的剩余方法和字段，请访问以下链接查看详细：
https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFCell.html

## XSSFCellStyle

这是在org.apache.poi.xssf.usermodel包的类。它将提供关于在电子表格的单元格中的内容的格式可能的信息。它也提供了用于修正该格式的选项。它实现了CellStyle接口。

### 字段摘要

下表列出了从CellStyle接口继承一些字段。

| 字段名称 | 字段描述 |
| --- | --- |
| ALIGN_CENTER | 中心对齐单元格内容 |
| ALIGN_CENTER_SELECTION | 中心选择水平对齐方式 |
| ALIGN_FILL | 单元格适应于内容的大小 |
| ALIGN_JUSTIFY | 适应单元格内容的宽度 |
| ALIGN_LEFT | 左对齐单元格内容 |
| ALIGN_RIGHT | 右对齐单元格内容 |
| BORDER_DASH_DOT | 使用破折号和点单元格样式 |
| BORDER_DOTTED | 用虚线边框的单元格样式 |
| BORDER_DASHED | 用虚线边框的单元格样式 |
| BORDER_THICK | 厚厚的边框单元格样式 |
| BORDER_THIN | 薄边框的单元格样式 |
| VERTICAL_BOTTOM | 对齐单元格内容的垂直下方 |
| VERTICAL_CENTER | 对齐单元格内容垂直居中 |
| VERTICAL_JUSTIFY | 对齐和垂直对齐的单元格内容 |
| VERTICAL_TOP | 顶部对齐为垂直对齐 |

## 类的构造函数

| S.No. | 构造函数及描述 |
| --- | --- |
| 1 | **XSSFCellStyle(int cellXfId, int cellStyleXfId, StylesTable stylesSource, ThemesTable theme)** 创建一个单元格样式，从所提供的部分 |
| 2 | **XSSFCellStyle(StylesTable stylesSource)** 创建一个空的单元样式 |

## 类方法

设置边框的类型为单元格的底部边界

| S.No | 方法及描述 |
| --- | --- |
| 1 | **setAlignment(short align)** 设置单元格为水平对齐的类型 |
| 2 | **setBorderBottom(short border)** |
| 3 | **setBorderColor(XSSFCellBorder.BorderSide side, XSSFColor color)** 选定的边框颜色 |
| 4 | **setBorderLeft(Short border)** 设置边界的类型单元格的左边框 |
| 5 | **setBorderRight(short border)** 设置边框的类型为单元格的右边界 |
| 6 | **setBorderTop(short border)** 设置边界的类型的单元上边框 |
| 7 | **setFillBackgroundColor(XSSFColor color)** 设置表示为XSSFColor值背景填充颜色。 |
| 8 | **setFillForegroundColor(XSSFColor color)** 设置表示为XSSFColor的值前景填充颜色。 |
| 9 | **setFillPattern(short fp)** 指定单元格的填充信息模式和纯色填充单元。 |
| 10 | **setFont(Font font)** 设置此样式的字体。 |
| 11 | **setRotation(short rotation)** 设置的旋转为在单元格中文本的程度。 |
| 12 | **setVerticalAlignment(short align)** 设置单元类型为垂直取向。 |

对于这个类剩下的方法和字段，通过以下链接：
https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFCellStyle.html

# HSSFColor

这是在org.apache.poi.hssf.util包的类。它提供了不同的颜色作为嵌套类。通常这些嵌套类是使用自己的索引来表示。它实现了Color接口。

## 嵌套类

所有嵌套类这个类是静态的，每个类都有其索引。这些嵌套色类用于单元格格式，如单元格内容，边框，前景和背景。下面列出了一些的嵌套类。

| S.No. | 类名（颜色） |
|---|---|
| 1 | HSSFColor.AQUA |
| 2 | HSSFColor.AUTOMATIC |
| 3 | HSSFColor.BLACK |
| 4 | HSSFColor.BLUE |
| 5 | HSSFColor.BRIGHT_GREEN |
| 6 | HSSFColor.BRIGHT_GRAY |
| 7 | HSSFColor.CORAL |
| 8 | HSSFColor.DARK_BLUE |
| 9 | HSSFColor.DARK_GREEN |
| 10 | HSSFColor.SKY_BLUE |
| 11 | HSSFColor.WHITE |
| 12 | HSSFColor.YELLOW |

### 类方法

这个类的只有一个方法是很重要的，并且用于获取索引值。

| S.No. | 方法和描述 |
|---|---|
| 1 | **getIndex()** 这种方法被用来获得一个嵌套类的索引值 |

对于其余的方法和嵌套类，请参阅以下链接：
https://poi.apache.org/apidocs/org/apache/poi/hssf/util/HSSFColor.html.

## XSSFColor

这是在org.apache.poi.xssf.usermodel包的类。它是用来表示在电子表格中的颜色。它实现了颜色的接口。下面列出的是它的一些方法和构造函数。

### 类的构造函数

| S.No. | Constructor and 描述 |
|---|---|
| 1 | **XSSFColor()** 创建XSSFColor的新实例。 |
| 2 | **XSSFColor(byte[] rgb)** 创建XSSFColor使用RGB的新实例。 |
| 3 | **XSSFColor(java.awt.Color clr)** 创建XSSFColor使用Color类从AWT包的新实例。 |

## 类方法

| S.No. | 方法和描述 |
|---|---|
| 1 | **setAuto(boolean auto)** 设置一个布尔值，表示ctColor是自动的，系统ctColor依赖。 |
| 2 | **setIndexed(int indexed)** 设置索引ctColor值系统ctColor。 |

对于其余的方法，请访问以下链接：
https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFColor.html.

# XSSFFont

这是在org.apache.poi.xssf.usermodel包的类。它实现了Font接口，因此它可以处理工作簿中不同的字体。

## 类的构造函数

| S.No. | 构造函数和描述 |
|---|---|
| 1 | **XSSFFont()** 创建一个新的XSSFont实例。 |

## 类方法

| S.No. | 方法和描述 |
|---|---|
| 1 | **setBold(boolean bold)** 设置“bold”属性的布尔值。 |
| 2 | **setColor(short color)** 设置索引颜色的字体。 |
| 3 | **setColor(XSSFColor color)** 设置为标准Alpha RGB颜色值的字体颜色。 |
| 4 | **setFontHeight(short height)** 设置在点的字体高度。 |
| 5 | **setFontName(java.lang.String name)** 设置字体的名称。 |
| 6 | **setItalic(boolean italic)** 设置“italic”属性一个布尔值。 |

对于其余的方法，通过以下链接：
https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFFont.html.

# XSSFHyperlink

这是在org.apache.poi.xssf.usermodel包的类。它实现了Hyperlink接口。它是用来连结设置为电子表格的单元格内容。

## 字段

属于此类的字段如下。这里，字段意味着使用超链接的类型。

| 字段 | 描述 |
|---|---|
| LINK_DOCUMENT | 用于连接任何其他文件 |
| LINK_EMAIL | 用于链接的电子邮件 |
| LINK_FILE | 用于以任何格式链接任何其他文件 |
| LINK_URL | 用来连接一个网页URL |

## 类方法

| S.No. | 方法及描述 |
|---|---|
| 1 | **setAddress(java.lang.String address)** 超链接地址。 |

对于其余的方法，请访问以下链接：
https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFHyperlink.htm
l

# XSSFCreationHelper

这是在org.apache.poi.xssf.usermodel包的类。它实现了CreationHelper接口。它被用作公式求值和设置超文本链接支持类。

## 类方法

| S.No. | 方法和描述 |
|---|---|
| 1 | **createFormulaEvaluator()** 创建一个XSSFFormulaEvaluator例如，结果计算公式的单元格的对象。 |
| 2 | **createHyperlink(int type)** Creates a new XSSFHyperlink. |

对于其余的方法，请参考以下链接：https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFCreationHelper.html.

# XSSFPrintSetup

这是在org.apache.poi.xsssf.usermodel包下的类。它实现了PrintSetup接口。它是用来设置打印页面大小，面积，选项和设置。

## 类方法

| S.No. | 方法及说明 |
|---|---|
| 1 | **setLandscape(boolean ls)** 设置一个布尔值，允许或阻止横向打印。 |
| 2 | **setLeftToRight(boolean ltor)** 设置是否向左走向右或自上而下的顺序，同时打印。 |
| 3 | **setPaperSize(short size)** 设置纸张尺寸。 |

对于其余的方法，请访问以下链接：https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFPrintSetup.html

# Apache POI工作簿 - POI教程

此处的术语“Workbook”指的Microsoft Excel文件。本章完成后，您将能够创建新的工作簿，并可以使用Java程序打开现有工作簿。

## 创建空白工作簿

下面简单的程序来创建一个空白Microsoft Excel工作簿。

```
import java.io.*;
import org.apache.poi.xssf.usermodel.*;
public class CreateWorkBook
{
   public static void main(String[] args)throws Exception
   {
      //Create Blank workbook
      XSSFWorkbook workbook = new XSSFWorkbook();
      //Create file system using specific name
      FileOutputStream out = new FileOutputStream(
      new File("createworkbook.xlsx"));
      //write operation workbook using file out object
      workbook.write(out);
      out.close();
      System.out.println("
      createworkbook.xlsx written successfully");
   }
}
```

让我们保存上面的Java代码为CreateWorkBook.java，然后编译并从命令提示符如下执行它：

```
$javac CreateWorkBook.java
$java CreateWorkBook
```

如果系统环境配置了POI 库，它会编译和执行，并生成一个名为 createworkbook.xlsx 在当前目录下的空白Excel文件并显示在命令提示符处键入以下输出。

```
createworkbook.xlsx written successfully
```

## 打开现有工作簿

使用下面的代码打开现有的工作簿。

```
import java.io.*;
import org.apache.poi.xssf.usermodel.*;
public class OpenWorkBook
{
   public static void main(String args[])throws Exception
   {
      File file = new File("openworkbook.xlsx");
      FileInputStream fIP = new FileInputStream(file);
      //Get the workbook instance for XLSX file
      XSSFWorkbook workbook = new XSSFWorkbook(fIP);
      if(file.isFile() && file.exists())
      {
         System.out.println(
         "openworkbook.xlsx file open successfully.");
      }
      else
      {
         System.out.println(
         "Error to open openworkbook.xlsx file.");
      }
   }
}
```

保存上面的Java代码为OpenWorkBook.java，然后编译并从命令提示符如下执行它：

```
$javac OpenWorkBook.java
$java OpenWorkBook
```

这将编译和执行生成以下输出。

```
openworkbook.xlsx file open successfully.
```

打开工作簿后，可以进行读取，并在上面写操作。

# Apache POI电子表格/Spreadsheet - POI教程

本章将介绍如何创建一个电子表格，并使用Java操纵它。电子表格是在Excel文件中的页面;它包含具有特定名称的行和列。

读完本章后，将能够创建一个电子表格，并能在其上执行读取操作。

## 创建电子表格

首先，让我们创建一个使用在前面的章节中讨论的引用的类的电子表格。按照前面的章节中，首先创建一个工作簿，然后我们就可以去，并创建一个表。

下面的代码片段用于创建电子表格。

```
//Create Blank workbook
XSSFWorkbook workbook = new XSSFWorkbook();
//Create a blank spreadsheet
XSSFSheet spreadsheet = workbook.createSheet("Sheet Name");
```

## 在电子表格的行

电子表格有一个网格布局。行和列被标识与特定的名称。该列标识字母和行用数字。

下面的代码片段用于创建一个行。

```
XSSFRow row = spreadsheet.createRow((short)1);
```

## 写入到电子表格

让我们考虑雇员数据的一个例子。这里的雇员数据给出以表格形式。

| Emp Id | Emp Name | 称号 |
|---|---|---|
| Tp01 | Gopal | Technical Manager |
| TP02 | Manisha | Proof Reader |
| Tp03 | Masthan | Technical Writer |
| Tp04 | Satish | Technical Writer |
| Tp05 | Krishna | Technical Writer |

以下代码是用来写上述数据到电子表格。

```
import java.io.File;
import java.io.FileOutputStream;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class Writesheet
{
   public static void main(String[] args) throws Exception
   {
      //Create blank workbook
      XSSFWorkbook workbook = new XSSFWorkbook();
      //Create a blank sheet
      XSSFSheet spreadsheet = workbook.createSheet(
      " Employee Info ");
      //Create row object
      XSSFRow row;
      //This data needs to be written (Object[])
      Map < String, Object[] > empinfo =
      new TreeMap < String, Object[] >();
      empinfo.put( "1", new Object[] {
      "EMP ID", "EMP NAME", "DESIGNATION" });
      empinfo.put( "2", new Object[] {
      "tp01", "Gopal", "Technical Manager" });
      empinfo.put( "3", new Object[] {
      "tp02", "Manisha", "Proof Reader" });
      empinfo.put( "4", new Object[] {
      "tp03", "Masthan", "Technical Writer" });
      empinfo.put( "5", new Object[] {
      "tp04", "Satish", "Technical Writer" });
      empinfo.put( "6", new Object[] {
      "tp05", "Krishna", "Technical Writer" });
      //Iterate over data and write to sheet
      Set < String > keyid = empinfo.keySet();
      int rowid = 0;
      for (String key : keyid)
      {
         row = spreadsheet.createRow(rowid++);
         Object [] objectArr = empinfo.get(key);
         int cellid = 0;
         for (Object obj : objectArr)
         {
            Cell cell = row.createCell(cellid++);
            cell.setCellValue((String)obj);
         }
      }
      //Write the workbook in file system
```

```
        FileOutputStream out = new FileOutputStream(
        new File("Writesheet.xlsx"));
        workbook.write(out);
        out.close();
        System.out.println(
        "Writesheet.xlsx written successfully" );
    }
}
```

上面的Java代码保存为Writesheet.java，然后并在命令提示符下编译运行，如下所示：

```
$javac Writesheet.java
$java Writesheet
```

这将编译和执行来生成一个Excel文件名为Writesheet.xlsx在当前目录中，在命令提示符处键入以下输出。

```
Writesheet.xlsx written successfully
```

Writesheet.xlsx文件的内容如下所示。

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | EMP ID | EMP NAME | DESIGNATION | | | | | | |
| 2 | tp01 | Gopal | Technical Manager | | | | | | |
| 3 | tp02 | Manisha | Proof Reader | | | | | | |
| 4 | tp03 | Masthan | Technical Writer | | | | | | |
| 5 | tp04 | Satish | Technical Writer | | | | | | |
| 6 | tp05 | Krishna | Technical Writer | | | | | | |
| 7 | | | | | | | | | |
| 8 | | | | | | | | | |
| 9 | | | | | | | | | |
| 10 | | | | | | | | | |
| 11 | | | | | | | | | |

## 从电子表格读取数据

让我们考虑上述excel文件命名Writesheet.xslx作为输入文件。注意下面的代码;它是用于从电子表格中读取数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class Readsheet
{
   static XSSFRow row;
   public static void main(String[] args) throws Exception
   {
      FileInputStream fis = new FileInputStream(
      new File("WriteSheet.xlsx"));
      XSSFWorkbook workbook = new XSSFWorkbook(fis);
      XSSFSheet spreadsheet = workbook.getSheetAt(0);
      Iterator < Row > rowIterator = spreadsheet.iterator();
      while (rowIterator.hasNext())
      {
         row = (XSSFRow) rowIterator.next();
         Iterator < Cell > cellIterator = row.cellIterator();
         while ( cellIterator.hasNext())
         {
            Cell cell = cellIterator.next();
            switch (cell.getCellType())
            {
               case Cell.CELL_TYPE_NUMERIC:
               System.out.print(
               cell.getNumericCellValue() + " \t\t " );
               break;
               case Cell.CELL_TYPE_STRING:
               System.out.print(
               cell.getStringCellValue() + " \t\t " );
               break;
            }
         }
         System.out.println();
      }
      fis.close();
   }
}
```

让我们把上面的代码保存在Readsheet.java文件，然后编译并在命令提示符下运行，如下所示：

```
$javac Readsheet.java
$java Readsheet
```

如果您的系统环境配置了POI库，它会编译和执行产生在命令提示符处键入以下输出。

```
EMP ID EMP NAME DESIGNATION
 tp01   Gopal    Technical Manager
 tp02   Manisha  Proof Reader
 tp03   Masthan  Technical Writer
 tp04   Satish   Technical Writer
 tp05   Krishna  Technical Writer
```

# Apache POI单元格/Cells - POI教程

输入到电子表格中的任何数据总是存储在一个单元中。我们使用的行和列的标签来识别单元格。本章介绍了如何使用Java编程操纵单元电子表格的数据。

## 创建一个单元格

需要创建一个单元之前创建一个行。行是什么？只不过是单元的集合。

下面的代码片段用于创建一个单元格。

```
//create new workbook
XSSFWorkbook workbook = new XSSFWorkbook();
//create spreadsheet with a name
XSSFSheet spreadsheet = workbook.createSheet("new sheet");
//create first row on a created spreadsheet
XSSFRow row = spreadsheet.createRow(0);
//create first cell on created row
XSSFCell cell = row.createCell(0);
```

## 单元格类型

单元格类型指定单元格是否可以包含字符串，数值，或公式。字符串单元不能持有数值和数值单元格无法容纳字符串。下面给出是单元格值和类型的语法。

| 单元格的值类型 | 类型语法 |
| --- | --- |
| Blank cell value | XSSFCell.CELL_TYPE_BLANK |
| Boolean cell value | XSSFCell.CELL.TYPE_BOOLEAN |
| Error cell value | XSSFCell.CELL_TYPE_ERROR |
| Numeric cell value | XSSFCell.CELL_TYPE_NUMERIC |
| String cell value | XSSFCell.CELL_TYPE_STRING |

以下代码是用于在电子表格创建不同类型的单元格。

```
import java.io.File;
import java.io.FileOutputStream;
import java.util.Date;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class TypesofCells
{
   public static void main(String[] args)throws Exception
   {
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook.createSheet("cell types");
      XSSFRow row = spreadsheet.createRow((short) 2);
      row.createCell(0).setCellValue("Type of Cell");
      row.createCell(1).setCellValue("cell value");
      row = spreadsheet.createRow((short) 3);
      row.createCell(0).setCellValue("set cell type BLANK");
      row.createCell(1);
      row = spreadsheet.createRow((short) 4);
      row.createCell(0).setCellValue("set cell type BOOLEAN");
      row.createCell(1).setCellValue(true);
      row = spreadsheet.createRow((short) 5);
      row.createCell(0).setCellValue("set cell type ERROR");
      row.createCell(1).setCellValue(XSSFCell.CELL_TYPE_ERROR );
      row = spreadsheet.createRow((short) 6);
      row.createCell(0).setCellValue("set cell type date");
      row.createCell(1).setCellValue(new Date());
      row = spreadsheet.createRow((short) 7);
      row.createCell(0).setCellValue("set cell type numeric" );
      row.createCell(1).setCellValue(20 );
      row = spreadsheet.createRow((short) 8);
      row.createCell(0).setCellValue("set cell type string");
      row.createCell(1).setCellValue("A String");
      FileOutputStream out = new FileOutputStream(
      new File("typesofcells.xlsx"));
      workbook.write(out);
      out.close();
      System.out.println(
      "typesofcells.xlsx written successfully");
   }
}
```

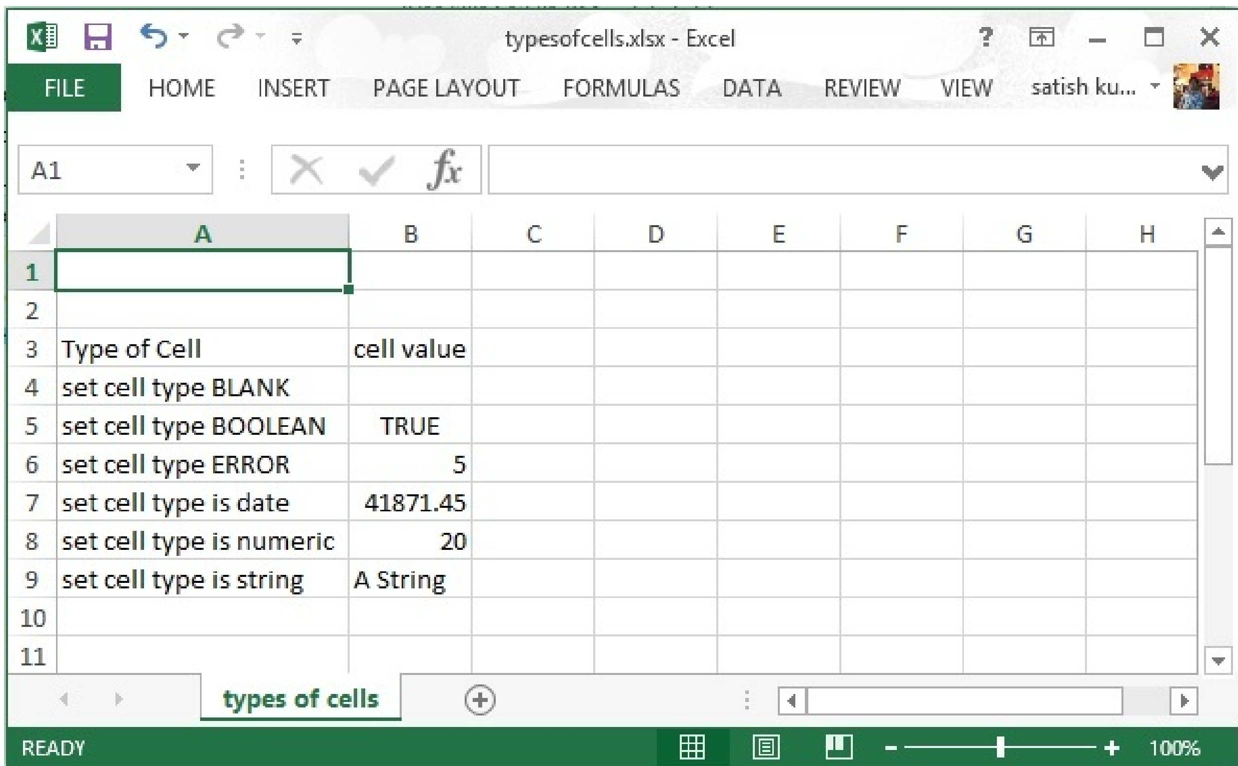
保存上面的代码到一个名为TypesofCells.java文件，编译并从命令提示符如下执行它。

```
$javac TypesofCells.java
$java TypesofCells
```

如果您的系统配置了POI库，那么它会编译和执行在当前目录中生成一个名为typesofcells.xlsx的Excel文件，并显示以下输出。

```
typesofcells.xlsx written successfully
```

typesofcells.xlsx文件如下所示。




# 单元格样式

在这里，可以学习如何做单元格格式，并采用不同的风格，如合并相邻的单元格，添加边框，设置单元格对齐方式和填充颜色。

以下代码是使用Java编程用于不同样式应用到单元格。

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.hssf.util.HSSFColor;
import org.apache.poi.ss.usermodel.IndexedColors;
import org.apache.poi.ss.util.CellRangeAddress;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class CellStyle
{
   public static void main(String[] args)throws Exception
```

```
   {
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook.createSheet("cellstyle");
      XSSFRow row = spreadsheet.createRow((short) 1);
      row.setHeight((short) 800);
      XSSFCell cell = (XSSFCell) row.createCell((short) 1);
      cell.setCellValue("test of merging");
      //MEARGING CELLS
      //this statement for merging cells
      spreadsheet.addMergedRegion(new CellRangeAddress(
      1, //first row (0-based)
      1, //last row (0-based)
      1, //first column (0-based)
      4 //last column (0-based)
      ));
      //CELL Alignment
      row = spreadsheet.createRow(5);
      cell = (XSSFCell) row.createCell(0);
      row.setHeight((short) 800);
      // Top Left alignment
      XSSFCellStyle style1 = workbook.createCellStyle();
      spreadsheet.setColumnWidth(0, 8000);
      style1.setAlignment(XSSFCellStyle.ALIGN_LEFT);
      style1.setVerticalAlignment(XSSFCellStyle.VERTICAL_TOP);
      cell.setCellValue("Top Left");
      cell.setCellStyle(style1);
      row = spreadsheet.createRow(6);
      cell = (XSSFCell) row.createCell(1);
      row.setHeight((short) 800);
      // Center Align Cell Contents
      XSSFCellStyle style2 = workbook.createCellStyle();
      style2.setAlignment(XSSFCellStyle.ALIGN_CENTER);
      style2.setVerticalAlignment(
      XSSFCellStyle.VERTICAL_CENTER);
      cell.setCellValue("Center Aligned");
      cell.setCellStyle(style2);
      row = spreadsheet.createRow(7);
      cell = (XSSFCell) row.createCell(2);
      row.setHeight((short) 800);
      // Bottom Right alignment
      XSSFCellStyle style3 = workbook.createCellStyle();
      style3.setAlignment(XSSFCellStyle.ALIGN_RIGHT);
      style3.setVerticalAlignment(
      XSSFCellStyle.VERTICAL_BOTTOM);
      cell.setCellValue("Bottom Right");
      cell.setCellStyle(style3);
      row = spreadsheet.createRow(8);
      cell = (XSSFCell) row.createCell(3);
      // Justified Alignment
      XSSFCellStyle style4 = workbook.createCellStyle();
      style4.setAlignment(XSSFCellStyle.ALIGN_JUSTIFY);
      style4.setVerticalAlignment(
      XSSFCellStyle.VERTICAL_JUSTIFY);
```

```
        cell.setCellValue("Contents are Justified in Alignment");
        cell.setCellStyle(style4);
        //CELL BORDER
        row = spreadsheet.createRow((short) 10);
        row.setHeight((short) 800);
        cell = (XSSFCell) row.createCell((short) 1);
        cell.setCellValue("BORDER");
        XSSFCellStyle style5 = workbook.createCellStyle();
        style5.setBorderBottom(XSSFCellStyle.BORDER_THICK);
        style5.setBottomBorderColor(
        IndexedColors.BLUE.getIndex());
        style5.setBorderLeft(XSSFCellStyle.BORDER_DOUBLE);
        style5.setLeftBorderColor(
        IndexedColors.GREEN.getIndex());
        style5.setBorderRight(XSSFCellStyle.BORDER_HAIR);
        style5.setRightBorderColor(
        IndexedColors.RED.getIndex());
        style5.setBorderTop(XSSFCellStyle.BIG_SPOTS);
        style5.setTopBorderColor(
        IndexedColors.CORAL.getIndex());
        cell.setCellStyle(style5);
        //Fill Colors
        //background color
        row = spreadsheet.createRow((short) 10 );
        cell = (XSSFCell) row.createCell((short) 1);
        XSSFCellStyle style6 = workbook.createCellStyle();
        style6.setFillBackgroundColor(
        HSSFColor.LEMON_CHIFFON.index );
        style6.setFillPattern(XSSFCellStyle.LESS_DOTS);
        style6.setAlignment(XSSFCellStyle.ALIGN_FILL);
        spreadsheet.setColumnWidth(1,8000);
        cell.setCellValue("FILL BACKGROUNG/FILL PATTERN");
        cell.setCellStyle(style6);
        //Foreground color
        row = spreadsheet.createRow((short) 12);
        cell = (XSSFCell) row.createCell((short) 1);
        XSSFCellStyle style7=workbook.createCellStyle();
        style7.setFillForegroundColor(HSSFColor.BLUE.index);
        style7.setFillPattern( XSSFCellStyle.LESS_DOTS);
        style7.setAlignment(XSSFCellStyle.ALIGN_FILL);
        cell.setCellValue("FILL FOREGROUND/FILL PATTERN");
        cell.setCellStyle(style7);
        FileOutputStream out = new FileOutputStream(
        new File("cellstyle.xlsx"));
        workbook.write(out);
        out.close();
        System.out.println("cellstyle.xlsx written successfully");
    }
}
```

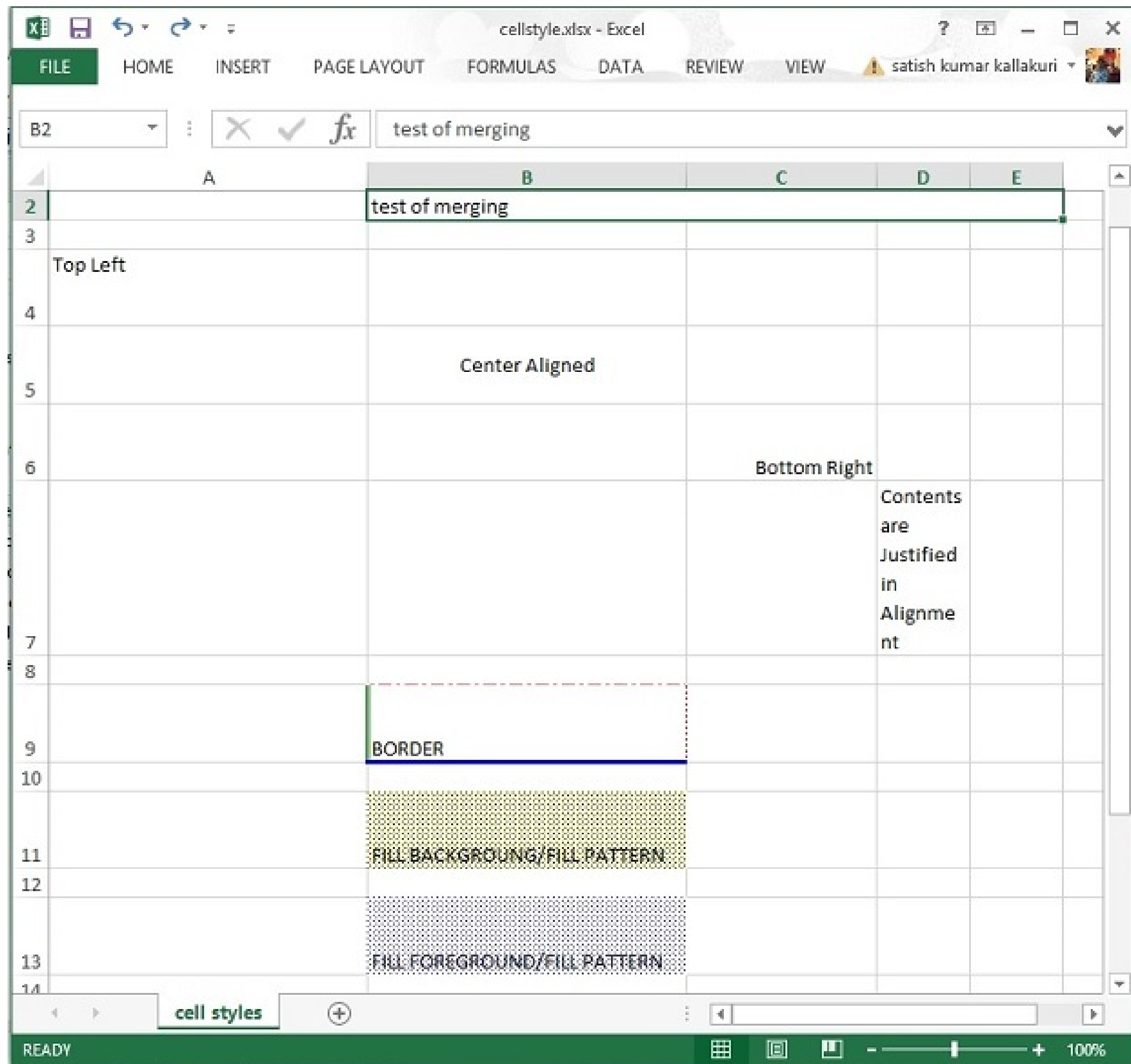
保存上面的代码在一个名为CellStyle.java文件，编译并从命令提示符如下执行它。

```
$javac CellStyle.java
$java CellStyle
```

它会生成一个名为cellstyle.xlsx在当前目录中的Excel文件并显示以下输出。

```
cellstyle.xlsx written successfully
```

cellstyle.xlsx文件如下所示。



# Apache POI字体/Fonts - POI教程

本章介绍如何设置不同的字体，应用样式，并在Excel电子表格中显示的方向不同角度的文字。

每个系统附带一个很大的字体如 Arial, Impact, Times New Roman,等字体集合也可以用新的字体更新，如果需要的话。同样也有各种风格，其中的字体可以显示，例如，粗体，斜体，下划线，删除线等。

## 字体和字体样式

下面的代码用于特定的字体和样式应用于一单元格的内容。

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.hssf.util.HSSFColor;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFFont;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class FontStyle
{
   public static void main(String[] args)throws Exception
   {
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook.createSheet("Fontstyle");
      XSSFRow row = spreadsheet.createRow(2);
      //Create a new font and alter it.
      XSSFFont font = workbook.createFont();
      font.setFontHeightInPoints((short) 30);
      font.setFontName("IMPACT");
      font.setItalic(true);
      font.setColor(HSSFColor.BRIGHT_GREEN.index);
      //Set font into style
      XSSFCellStyle style = workbook.createCellStyle();
      style.setFont(font);
      // Create a cell with a value and set style to it.
      XSSFCell cell = row.createCell(1);
      cell.setCellValue("Font Style");
      cell.setCellStyle(style);
      FileOutputStream out = new FileOutputStream(
      new File("fontstyle.xlsx"));
      workbook.write(out);
      out.close();
      System.out.println(
      "fontstyle.xlsx written successfully");
   }
}
```

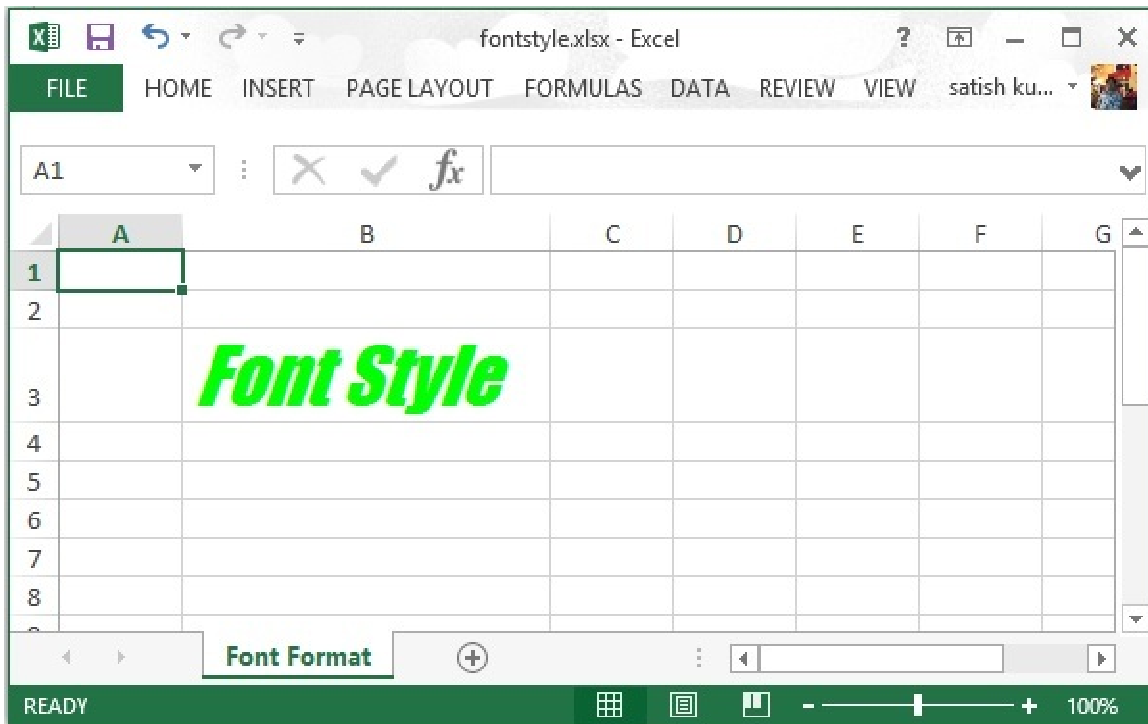
让我们保存上面的代码在一个名为FontStyle.java文件。从命令提示符编译并执行它如下。

```
$javac FontStyle.java
$java FontStyle
```

它生成一个名为fontstyle.xlsx在当前目录中的Excel文件并显示在命令提示符处键入以下输出。

```
fontstyle.xlsx written successfully
```

fontstyle.xlsx文件如下所示。




## 文字方向

在这里，可以学习如何设置不同角度的文本方向。通常单元格的内容以水平方式显示，由左到右，并在00角;但是可以使用下面的代码来旋转文本的方向(如果需要的话)。

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class TextDirection
{
   public static void main(String[] args)throws Exception
   {
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook.createSheet(
      "Text direction");
      XSSFRow row = spreadsheet.createRow(2);
      XSSFCellStyle myStyle = workbook.createCellStyle();
      myStyle.setRotation((short) 0);
      XSSFCell cell = row.createCell(1);
      cell.setCellValue("0D angle");
      cell.setCellStyle(myStyle);
      //30 degrees
      myStyle=workbook.createCellStyle();
```

```
        myStyle.setRotation((short) 30);
        cell = row.createCell(3);
        cell.setCellValue("30D angle");
        cell.setCellStyle(myStyle);
        //90 degrees
        myStyle=workbook.createCellStyle();
        myStyle.setRotation((short) 90);
        cell = row.createCell(5);
        cell.setCellValue("90D angle");
        cell.setCellStyle(myStyle);
        //120 degrees
        myStyle=workbook.createCellStyle();
        myStyle.setRotation((short) 120);
        cell = row.createCell(7);
        cell.setCellValue("120D angle");
        cell.setCellStyle(myStyle);
        //270 degrees
        myStyle = workbook.createCellStyle();
        myStyle.setRotation((short) 270);
        cell = row.createCell(9);
        cell.setCellValue("270D angle");
        cell.setCellStyle(myStyle);
        //360 degrees
        myStyle=workbook.createCellStyle();
        myStyle.setRotation((short) 360);
        cell = row.createCell(12);
        cell.setCellValue("360D angle");
        cell.setCellStyle(myStyle);
        FileOutputStream out = new FileOutputStream(
        new File("textdirection.xlsx"));
        workbook.write(out);
        out.close();
        System.out.println(
        "textdirection.xlsx written successfully");
    }
}
```

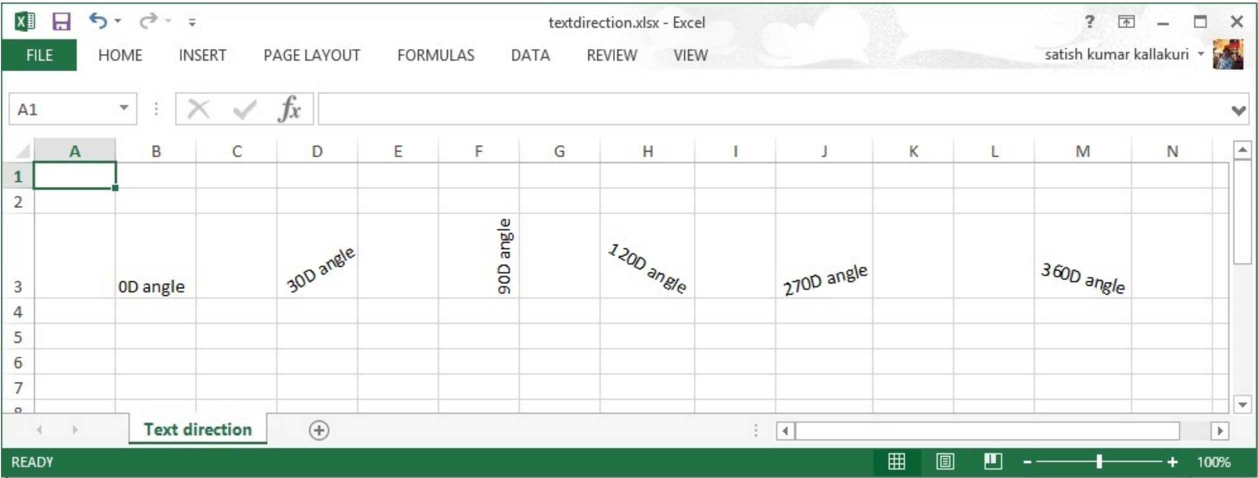
保持TextDirectin.java文件上面的代码，然后编译并从命令提示符如下执行它。

```
$javac TextDirection.java
$java TextDirection
```

这将编译和执行，以生成一个名为textdirection.xlsx在当前目录中的Excel文件并显示在命令提示符处键入以下输出。

```
textdirection.xlsx written successfully
```

textdirection.xlsx文件如下所示。


textdirection.xlsx - Excel
FILE
HOME
INSERT
PAGE LAYOUT
FORMULAS
DATA
REVIEW
VIEW
satish kumar kallakuri
A1
0D angle
30D angle
90D angle
120D angle
270D angle
360D angle
Text direction
READY
100%

# Apache POI公式 - POI教程

本章将介绍如何使用Java编程应用不同单元公式的过程。 Excel应用程序的基本目的是通过应用公式就可以保持数值数据。

在公式中，我们通过动态值，或在Excel工作表中的值的位置。在执行这个公式，就会得到想要的结果。下表列出了常用的在Excel中的几个基本公式。

| 操作 | 语法 |
| --- | --- |
| 添加多个数值 | = SUM(Loc1:Locn) or = SUM(n1,n2,) |
| 计数 | = COUNT(Loc1:Locn) or = COUNT(n1,n2,) |
| 两个数的幂 | = POWER(Loc1,Loc2) or = POWER(number, power) |
| 多个数的最大值 | = MAX(Loc1:Locn) or = MAX(n1,n2,) |
| 乘积 | = PRODUCT(Loc1:Locn) or = PRODUCT(n1,n2,) |
| 阶乘 | = FACT(Locn) or = FACT(number) |
| 绝对数字 | = ABS(Locn) or = ABS(number) |
| 今天的日期 | =TODAY() |
| 转换成小写 | = LOWER(Locn) or = LOWER(text) |
| 平方根 | = SQRT(locn) or = SQRT(number) |

以下代码用于公式添加至单元格，并执行它。

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class Formula
{
   public static void main(String[] args)throws Exception
   {
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook.createSheet("formula");
      XSSFRow row = spreadsheet.createRow(1);
      XSSFCell cell = row.createCell(1);
      cell.setCellValue("A =" );
      cell = row.createCell(2);
      cell.setCellValue(2);
      row = spreadsheet.createRow(2);
      cell = row.createCell(1);
```

```
        cell.setCellValue("B =");
        cell = row.createCell(2);
        cell.setCellValue(4);
        row = spreadsheet.createRow(3);
        cell = row.createCell(1);
        cell.setCellValue("Total =");
        cell = row.createCell(2);
        // Create SUM formula
        cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
        cell.setCellFormula("SUM(C2:C3)" );
        cell = row.createCell(3);
        cell.setCellValue("SUM(C2:C3)");
        row = spreadsheet.createRow(4);
        cell = row.createCell(1);
        cell.setCellValue("POWER =");
        cell=row.createCell(2);
        // Create POWER formula
        cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
        cell.setCellFormula("POWER(C2,C3)");
        cell = row.createCell(3);
        cell.setCellValue("POWER(C2,C3)");
        row = spreadsheet.createRow(5);
        cell = row.createCell(1);
        cell.setCellValue("MAX =");
        cell = row.createCell(2);
        // Create MAX formula
        cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
        cell.setCellFormula("MAX(C2,C3)");
        cell = row.createCell(3);
        cell.setCellValue("MAX(C2,C3)");
        row = spreadsheet.createRow(6);
        cell = row.createCell(1);
        cell.setCellValue("FACT =");
        cell = row.createCell(2);
        // Create FACT formula
        cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
        cell.setCellFormula("FACT(C3)");
        cell = row.createCell(3);
        cell.setCellValue("FACT(C3)");
        row = spreadsheet.createRow(7);
        cell = row.createCell(1);
        cell.setCellValue("SQRT =");
        cell = row.createCell(2);
        // Create SQRT formula
        cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
        cell.setCellFormula("SQRT(C5)");
        cell = row.createCell(3);
        cell.setCellValue("SQRT(C5)");
        workbook.getCreationHelper()
        .createFormulaEvaluator()
        .evaluateAll();
        FileOutputStream out = new FileOutputStream(
        new File("formula.xlsx"));
```

```
        workbook.write(out);
        out.close();
        System.out.println("fromula.xlsx written successfully");
    }
}
```

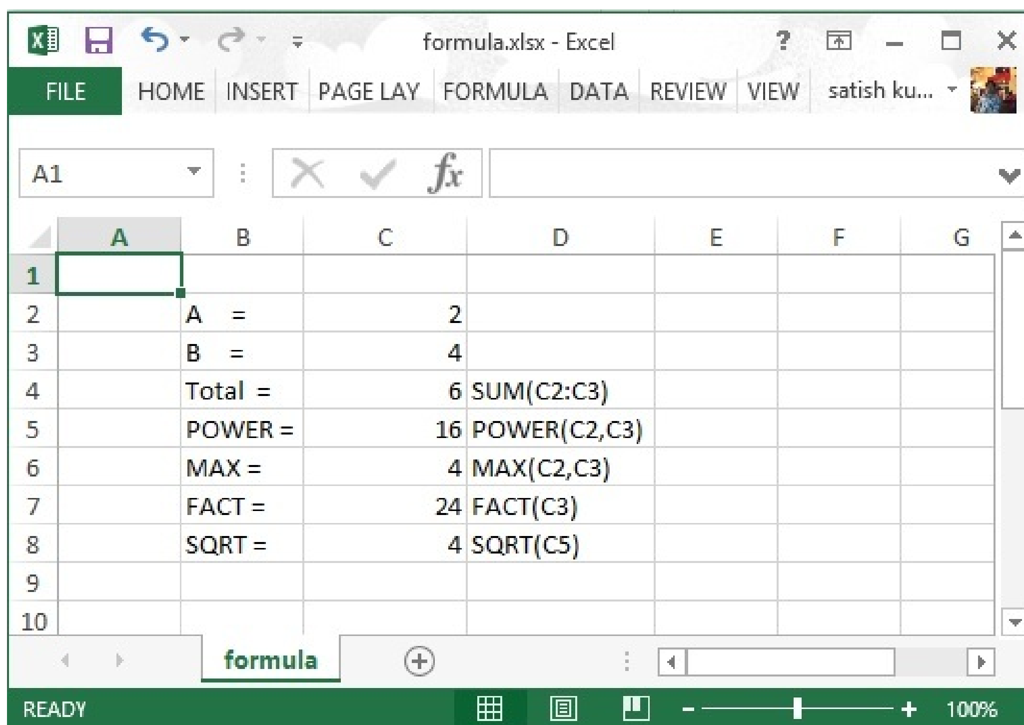
保存上面的代码到文件Formula.java，然后编译并从命令提示符如下执行它。

```
$javac Formula.java
$java Formula
```

它会生成一个名为formula.xlsx在当前目录中的Excel文件并显示在命令提示符处键入以下输出。

```
fromula.xlsx written successfully
```

formula.xlsx文件如下所示。



# Apache POI超链接 - POI教程

本章介绍了如何为超链接添加到内容的单元格。超链接通常被用来访问任何网站的网址，电子邮件或外部文件。

下面的代码演示如何创建单元格的超链接。

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.common.usermodel.Hyperlink;
import org.apache.poi.hssf.util.HSSFColor;
import org.apache.poi.ss.usermodel.CreationHelper;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFFont;
import org.apache.poi.xssf.usermodel.XSSFHyperlink;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class HyperlinkEX
{
   public static void main(String[] args) throws Exception
   {
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook
      .createSheet("Hyperlinks");
      XSSFCell cell;
      CreationHelper createHelper = workbook
      .getCreationHelper();
      XSSFCellStyle hlinkstyle = workbook.createCellStyle();
      XSSFFont hlinkfont = workbook.createFont();
      hlinkfont.setUnderline(XSSFFont.U_SINGLE);
      hlinkfont.setColor(HSSFColor.BLUE.index);
      hlinkstyle.setFont(hlinkfont);
      //URL Link
      cell = spreadsheet.createRow(1)
      .createCell((short) 1);
      cell.setCellValue("URL Link");
      XSSFHyperlink link = (XSSFHyperlink)createHelper
      .createHyperlink(Hyperlink.LINK_URL);
      link.setAddress("http://www.yiibai.com/" );
      cell.setHyperlink((XSSFHyperlink) link);
      cell.setCellStyle(hlinkstyle);
      //Hyperlink to a file in the current directory
      cell = spreadsheet.createRow(2)
      .createCell((short) 1);
      cell.setCellValue("File Link");
      link = (XSSFHyperlink)createHelper
      .createHyperlink(Hyperlink.LINK_FILE);
      link.setAddress("cellstyle.xlsx");
```

```
        cell.setHyperlink(link);
        cell.setCellStyle(hlinkstyle);
        //e-mail link
        cell = spreadsheet.createRow(3)
        .createCell((short) 1);
        cell.setCellValue("Email Link");
        link = (XSSFHyperlink)createHelper
        .createHyperlink(Hyperlink.LINK_EMAIL);
        link.setAddress(
        "mailto:contact@yiibai.com?"
        +"subject=Hyperlink");
        cell.setHyperlink(link);
        cell.setCellStyle(hlinkstyle);
        FileOutputStream out = new FileOutputStream(
        new File("hyperlink.xlsx"));
        workbook.write(out);
        out.close();
        System.out.println("hyperlink.xlsx written successfully");
    }
}
```

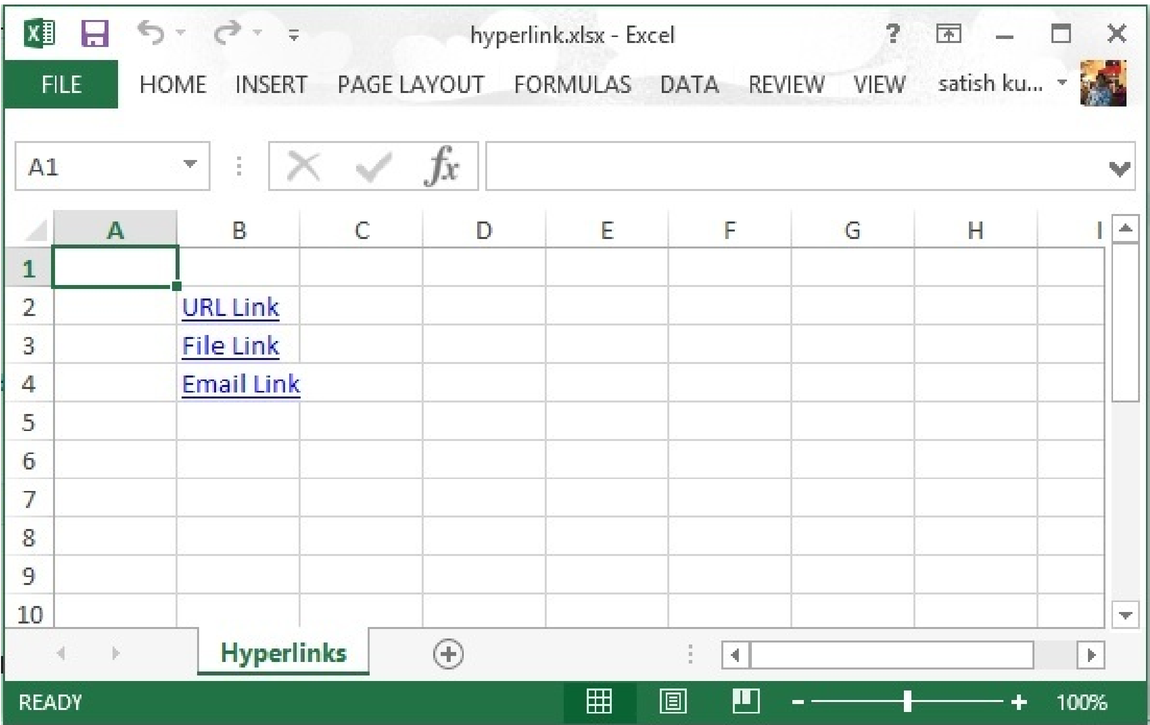
保存上面的代码到文件HyperlinkEX.java。并从命令提示符编译执行它如下。

```
$javac HyperlinkEX.java
$java HyperlinkEX
```

它会生成一个名为hyperlink.xlsx在当前目录中的Excel文件并显示在命令提示符处输出：

```
hyperlink.xlsx written successfully
```

hyperlink.xlsx文件如下所示。


hyperlink.xlsx - Excel
FILE
HOME
INSERT
PAGE LAYOUT
FORMULAS
DATA
REVIEW
VIEW
satish ku...
A1
URL Link
File Link
Email Link
Hyperlinks
READY
100%

# Apache POI打印区域 - POI教程

本章介绍了如何在电子表格中设置打印区域。通常打印区域从左上角到Excel电子表格右下角。打印区域可根据要求进行定制。它意味着可以从整个电子表格打印单元的特定范围，自定义的纸张大小，用网格线打印的内容接通等

以下代码是用来在电子表格中设置打印区域。

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFPrintSetup;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class PrintArea
{
   public static void main(String[] args)throws Exception
   {
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook
      .createSheet("Print Area");
      //set print area with indexes
      workbook.setPrintArea(
      0, //sheet index
      0, //start column
      5, //end column
      0, //start row
      5 //end row
      );
      //set paper size
      spreadsheet.getPrintSetup().setPaperSize(
      XSSFPrintSetup.A4_PAPERSIZE);
      //set display grid lines or not
      spreadsheet.setDisplayGridlines(true);
     //set print grid lines or not
     spreadsheet.setPrintGridlines(true);
     FileOutputStream out = new FileOutputStream(
     new File("printarea.xlsx"));
     workbook.write(out);
     out.close();
     System.out.println("printarea.xlsx written successfully");
   }
}
```

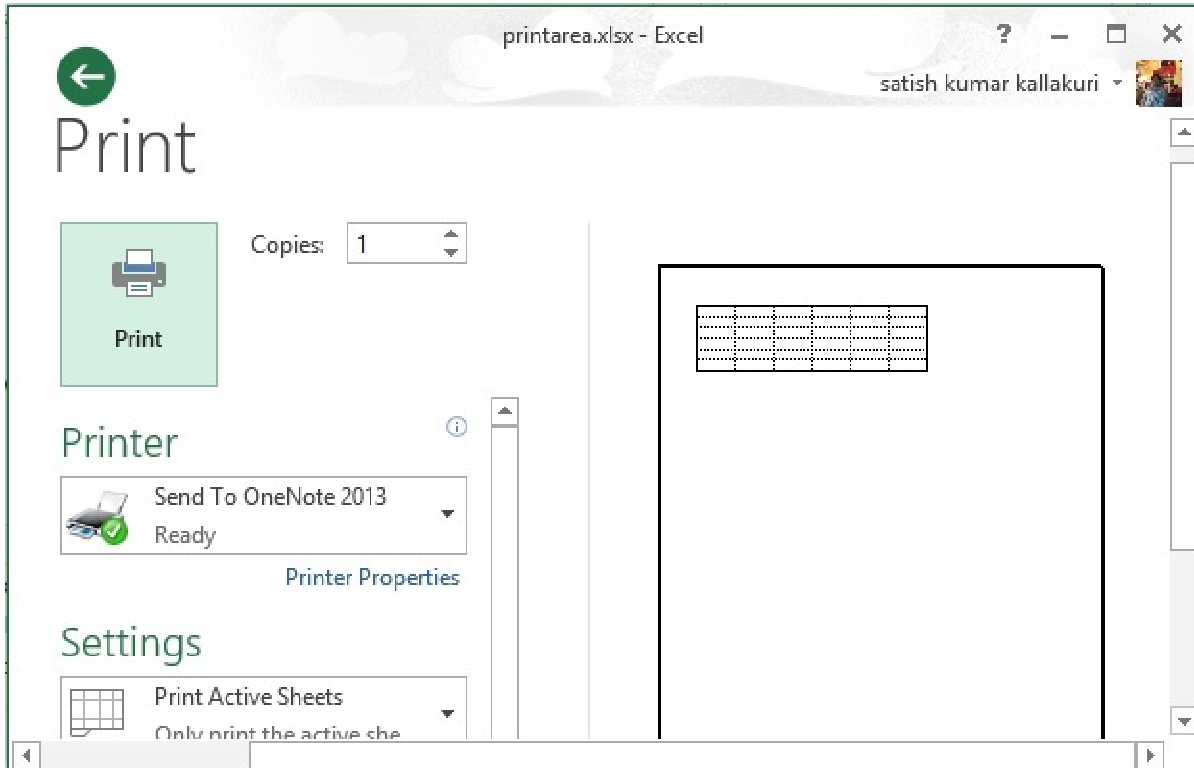
让我们保存了上面的代码为PrintArea.java。编译并从命令提示符执行它如下。

```
$javac PrintArea.java
$java PrintArea
```

它会生成一个名为printarea.xlsx在当前目录下的文件，并显示在命令提示符处输出以下。

```
printarea.xlsx written successfully
```

在上面的代码中，我们还没有添加任何单元格值。因此printarea.xlsx是一个空白文件。但是可以在下图的打印预览显示网格线打印区域查看。



# Apache POI数据库 - POI教程

本章介绍了POI库与数据库的交互方式。有了JDBC帮助，可以从数据库中检索数据并插入数据来使用POI库电子表格。让我们考虑SQL操作MySQL数据库。

## 写入数据库

让我们假设数据表是 emp_tbl 存有雇员信息是从MySQL数据库 test 中检索。

| EMP ID | EMP NAME | DEG | SALARY | DEPT |
|---|---|---|---|---|
| 1201 | Gopal | Technical Manager | 45000 | IT |
| 1202 | Manisha | Proof reader | 45000 | Testing |
| 1203 | Masthanvali | Technical Writer | 45000 | IT |
| 1204 | Kiran | Hr Admin | 40000 | HR |
| 1205 | Kranthi | Op Admin | 30000 | |

使用下面的代码从数据库中检索数据，并插入到同一个电子表格。

```
import java.io.File;
import java.io.FileOutputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class ExcelDatabase
{
   public static void main(String[] args) throws Exception
   {
      Class.forName("com.mysql.jdbc.Driver");
      Connection connect = DriverManager.getConnection(
      "jdbc:mysql://localhost:3306/test" ,
      "root" ,
      "root"
      );
      Statement statement = connect.createStatement();
      ResultSet resultSet = statement
      .executeQuery("select * from emp_tbl");
      XSSFWorkbook workbook = new XSSFWorkbook();
      XSSFSheet spreadsheet = workbook
```

```
        .createSheet("employe db");
        XSSFRow row=spreadsheet.createRow(1);
        XSSFCell cell;
        cell=row.createCell(1);
        cell.setCellValue("EMP ID");
        cell=row.createCell(2);
        cell.setCellValue("EMP NAME");
        cell=row.createCell(3);
        cell.setCellValue("DEG");
        cell=row.createCell(4);
        cell.setCellValue("SALARY");
        cell=row.createCell(5);
        cell.setCellValue("DEPT");
        int i=2;
        while(resultSet.next())
        {
           row=spreadsheet.createRow(i);
           cell=row.createCell(1);
           cell.setCellValue(resultSet.getInt("eid"));
           cell=row.createCell(2);
           cell.setCellValue(resultSet.getString("ename"));
           cell=row.createCell(3);
           cell.setCellValue(resultSet.getString("deg"));
           cell=row.createCell(4);
           cell.setCellValue(resultSet.getString("salary"));
           cell=row.createCell(5);
           cell.setCellValue(resultSet.getString("dept"));
           i++;
        }
        FileOutputStream out = new FileOutputStream(
        new File("exceldatabase.xlsx"));
        workbook.write(out);
        out.close();
        System.out.println(
        "exceldatabase.xlsx written successfully");
     }
  }
```

让我们保存了上面的代码为ExcelDatabase.java。编译并从命令提示符执行它如下。

```
$javac ExcelDatabase.java
$java ExcelDatabase
```

它会生成一个名为exceldatabase.xlsx在当前目录中的Excel文件并显示在命令提示符处输出以下。

```
exceldatabase.xlsx written successfully
```

exceldatabase.xlsx文件如下所示。

exceldatabase.xlsx - Excel

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | |
| 2 | | EMP ID | EMP NAME | DEG | SALARY | DEPT | | |
| 3 | | 1201 | gopal | technical manager | 45000 | IT | | |
| 4 | | 1202 | manisha | proof reader | 45000 | testing | | |
| 5 | | 1203 | masthanvalli | developer | 40000 | IT | | |
| 6 | | 1204 | krian | Hr Admin | 40000 | HR | | |
| 7 | | 1205 | kranthi | Op Admin | 30000 | ADMIN | | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |
| 11 | | | | | | | | |

employe db

# AWT 教程

JAVA提供了一组丰富的平台无关的方式来创建图形用户界面的库。在这篇文章中，我们将看看在AWT（抽象窗口工具包）。

## 读者

本教程是专为愿意学习JAVA的GUI编程软件专业人员提供简单轻松的入门步骤。本教程将让你对JAVA的GUI编程有一个概念的理解，并完成本教程后，您可以把自己的专业知识水平较高的专业技术水平。

## 必备条件

继续本教程之前，你应该有Java编程语言的一个基本的了解，文本编辑器和执行程序等

# AWT 概述介绍 - AWT

## 图形用户界面

图形用户界面（GUI）提供用户交互通过一些图形组件。例如，我们的基础操作系统，还提供了GUI，通过窗口，框架，面板，按钮，文本区域，列表框，组合框，标签，复选框等，这些都被称为组件。使用这些组件，我们可以创建一个应用程序交互的用户界面。

完全是基于GUI提供的结果为最终用户响应唤起events.GUI事件。例如点击一个按钮，关闭窗口，打开一个窗口，输入的东西在一个textarea等，这些活动被称为events.GUI使得它更容易为最终用户使用的应用程序。这也使得他们有趣。

## 基本名词术语

| 术语 | 描述 |
| --- | --- |
| Component | 组件是一个对象，具有可以显示在屏幕上，并且可以与用户交互的图形表示。对于例子按钮，复选框，列表和滚动条的图形用户界面。 |
| Container | 容器对象中是一个组件，它可以含有其它成分。添加到容器中的组件列表中的被跟踪。列表中的顺序将定义组件的堆叠顺序从前到后的容器内。如果未指定索引，将一个组件添加到容器中时，它会被添加到列表的末尾。 |
| Panel | 面板提供了空间，在其中一个应用程序可以连接任何其他组件，包括其他面板。 |
| Window | 窗口是显示在屏幕上的一个矩形区域。在不同的窗口中，我们可以执行不同的程序，并显示不同的数据。窗口为我们提供了多任务环境。一个窗口，必须有一个框，对话框，或定义为它的主人时，它的构造的另一个窗口。 |
| Frame | Frame是带有标题和边框的顶层窗口。帧的大小，包括任何指定为边界的区域。帧封装窗口。它有标题栏，菜单栏，边框和调整大小角落。 |
| Canvas | Canvas组件表示空白屏幕上的应用程序可以绘制矩形区域。应用程序也可以使用Canvas组件，空白区域捕获输入事件。 |

## 基于GUI的应用实例

以下是一些基于图形用户界面的应用程序的例子。

- 自动取款机 (ATM)
- 航空票务系统
- 在火车站的信息服务亭
- 移动应用程序
- 导航系统

## 图形用户界面较字符界面的优点

- GUI提供图形化的图标进行互动，而CUI（字符用户界面）提供简单的基于文本的接口。
- 图形用户界面，使应用更多的娱乐性和趣味性，另一方面CUI没有（或不多）。
- GUI提供点击和执行环境，而在CUI我们每次都要输入命令任务。
- 新的用户可以很容易地与图形用户界面交互的视觉指示器，但很难在字符的用户界面。
- GUI提供了很多的文件系统的控制和操作系统，而在CUI，必须使用命令很难记住。
- Windows概念在GUI中允许用户查看，操纵和控制的多个应用程序一次，而在CUI用户可以在同一时间控制一个任务。
- GUI提供了多任务环境中，使CUI也没有，但CUI不提供的GUI容易操作。
- 使用图形用户界面，它更容易控制和导航的操作系统，它在命令的用户界面变得非常慢。 GUI可以轻松定制。

# AWT 环境设置 - AWT

本节将指导如何下载并设置Java在您的机器上。请按照下列步骤来设置环境。

Java SE 是免费提供下载的链接Download Java. 所以，根据您的操作系统下载一个版本。

按照说明下载java和运行.exe文件在你的机器上安装Java。一旦在你的机器上安装了Java，就需要设置环境变量指向正确的安装目录：

## 设置窗口XP/WIN7路径：

假设您已经安装了Java在c:Program Filesjavajdk 目录:

- 右键点击“我的电脑”，选择“属性”。
- 点击“环境变量”按钮下的“高级”选项卡。
- 现在改变的“路径”变量，因此，它也包含Java可执行文件的路径。例如，如果路径当前设置为'C:WINDOWSSYSTEM32',然后可更改您的路径'C:WINDOWSSYSTEM32;c:Program Filesjavajdkin'.

## 适用于Linux，UNIX，Solaris和FreeBSD的路径：

应设置环境变量PATH指向已经安装Java二进制文件。如果这样做有困难，请参阅shell文件。

例如，如果使用bash作为shell，添加以下行到结束行'.bashrc: export PATH=/path/to/java:$PATH'

## 流行的Java编辑器：

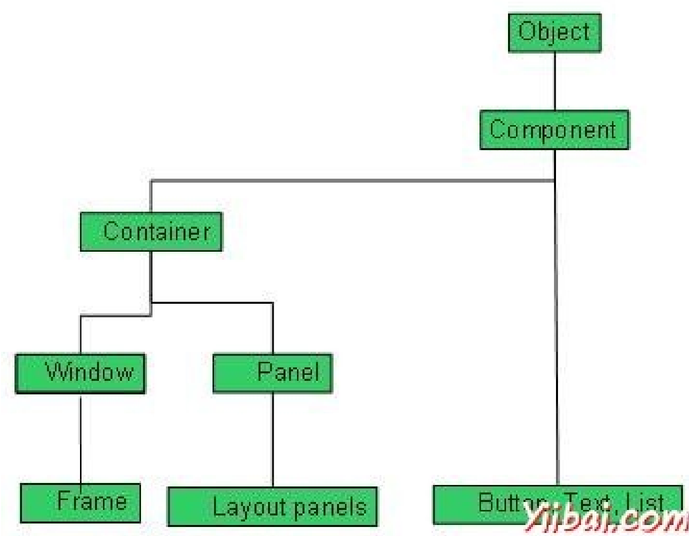
编写Java程序，您将需要一个文本编辑器。有更复杂的IDE，可以在市场上。但现在，可以考虑以下几种：

- Notepad : 在Windows机器上，你可以使用任何简单的文本编辑器（如记事本）（本教程推荐），TextPad。
- Netbeans : 是一个Java IDE，是开源和免费，可以直接从网上下载 http://www.netbeans.org/index.html.
- Eclipse : 也是一个Java IDE Eclipse开源社区开发的，可以直接从网上下载 http://www.eclipse.org/.

# AWT控件(Controls) - AWT

每一个用户界面参考主要有以下三个方面：

- UI 元素: 有核心可视化元素最终用户看到并与之交互。 GWT提供了一个巨大的名单变化复杂，我们将在本教程从基本的广泛使用和常见的元素。
- 布局: 他们定义的UI元素如何应安排在屏幕上，并提供一个最终的外观和感觉的GUI（图形用户界面）。在“布局”一章，这部分将被覆盖。
- 行为: 这些事件发生时，与用户交互的UI元素。这部分将被覆盖在"事件处理"一章。




每个AWT控件继承组件类的属性。

| Sr. No. | 控件与描述 |
|---|---|
| 1 | Component 组件是一个抽象的超类的GUI控件，并用图形表示，它代表了一个对象。 |

## AWT的UI元素：

以下是常用的控件列表而设计的图形用户界面使用AWT。

| Sr.No. | 控件与描述 |
| --- | --- |
| 1 | Label 标签对象是在容器中放置文本的一个组成部分。 |
| 2 | Button 该类创建标记的按钮。 |
| 3 | Check Box 复选框是一个图形化的组件，它可以在一个（真）或关闭（假）的状态。 |
| 4 | Check Box Group CheckboxGroup 类用于分组组复选框。 |
| 5 | List List组件为用户提供了一个滚动的文本项列表。 |
| 6 | Text Field TextField对象是一个文本组件，它允许编辑的单行文本。 |
| 7 | Text Area 一个textarea对象是一个文本组件，它允许编辑的多行文本。 |
| 8 | Choice 一个选择控制用于显示弹出菜单选择。所选选项将显示在顶部的菜单。 |
| 9 | Canvas Canvas控件代表一个矩形区域，应用程序可以画的东西，或者可以由用户创建的接收输入。 |
| 10 | Image 图像控制是表示图形图像的所有图像类的超类。 |
| 11 | Scroll Bar 代表一个滚动条的滚动条控制组件以让用户选择的值来自范围。 |
| 12 | Dialog 一个对话框控制是一个顶层窗口，采取某种形式的用户输入的标题和边框。 |
| 13 | File Dialog 一个FileDialog的控制代表一个对话框窗口，用户可以选择一个文件。 |

# AWT Component类 - AWT

## 介绍

Component 类 是一个非菜单用户界面控件，是AWT一个抽象基类。组件表示图形表示的对象。

## 类的声明

以下是声明的java.awt.Component类：

```
public abstract class Component
   extends Object
      implements ImageObserver, MenuContainer, Serializable
```

## 字段域

以下是 java.awt.Component 类的字段:

- static float BOTTOM_ALIGNMENT -- 易于使用的常数getAlignmentY.
- static float CENTER_ALIGNMENT -- 易于使用的常数 getAlignmentY 和 getAlignmentX。
- static float LEFT_ALIGNMENT -- 易于使用常数getAlignmentX。
- static float RIGHT_ALIGNMENT -- 易于使用常数getAlignmentX。
- static float TOP_ALIGNMENT -- 易于使用的常数 getAlignmentY().

## 类的构造函数

| S.N. | 构造函数与说明 |
|---|---|
| 1 | **protected Component()** 这将创建一个新的组件。 |

## 类方法

| S.N. | 方法和说明 |
|---|---|
| | **boolean action(Event evt, Object what)** Deprecated. As of JDK |

| | |
|---|---|
| 1 | version 1.1, should register this component as ActionListener on component which fires action events. |
| 2 | **void add(PopupMenu popup)** Adds the specified popup menu to the component. |
| 3 | **void addComponentListener(ComponentListener l)** Adds the specified component listener to receive component events from this component. |
| 4 | **void addFocusListener(FocusListener l)** Adds the specified focus listener to receive focus events from this component when this component gains input focus. |
| 5 | **void addHierarchyBoundsListener(HierarchyBoundsListener l)** Adds the specified hierarchy bounds listener to receive hierarchy bounds events from this component when the hierarchy to which this container belongs changes. |
| 6 | **void addHierarchyListener(HierarchyListener l)** Adds the specified hierarchy listener to receive hierarchy changed events from this component when the hierarchy to which this container belongs changes. |
| 7 | **void addInputMethodListener(InputMethodListener l)** Adds the specified input method listener to receive input method events from this component. |
| 8 | **void addKeyListener(KeyListener l)** Adds the specified key listener to receive key events from this component. |
| 9 | **void addMouseListener(MouseListener l)** Adds the specified mouse listener to receive mouse events from this component. |
| 10 | **void addMouseMotionListener(MouseMotionListener l)** Adds the specified mouse motion listener to receive mouse motion events from this component. |
| 11 | **void addMouseWheelListener(MouseWheelListener l)** Adds the specified mouse wheel listener to receive mouse wheel events from this component. |
| 12 | **void addNotify()** Makes this Component displayable by connecting it to a native screen resource. |
| 13 | **void addPropertyChangeListener(PropertyChangeListener listener)** Adds a PropertyChangeListener to the listener list. |
| 14 | **void addPropertyChangeListener(String propertyName, PropertyChangeListener listener)** Adds a PropertyChangeListener to the listener list for a specific property. |
| | **void applyComponentOrientation(ComponentOrientation** |

| | |
|---|---|
| 15 | **orientation)** Sets the ComponentOrientation property of this component and all components contained within it. |
| 16 | **boolean areFocusTraversalKeysSet(int id)** Returns whether the Set of focus traversal keys for the given focus traversal operation has been explicitly defined for this Component. |
| 17 | **int checkImage(Image image, ImageObserver observer)** Returns the status of the construction of a screen representation of the specified image. |
| 18 | **int checkImage(Image image,int width,int height, ImageObserver observer)** Returns the status of the construction of a screen representation of the specified image. |
| 19 | **boolean contains(int x,int y)** Checks whether this component "contains" the specified point, where x and y are defined to be relative to the coordinate system of this component. |
| 20 | **boolean contains(Point p)** Checks whether this component "contains" the specified point, where the point's x and y coordinates are defined to be relative to the coordinate system of this component. |
| 21 | **Image createImage(ImageProducer producer)** Creates an image from the specified image producer. |
| 22 | **Image createImage(int width,int height)** Creates an off-screen drawable image to be used for double buffering. |
| 23 | **VolatileImage createVolatileImage(int width,int height)** Creates a volatile off-screen drawable image to be used for double buffering. |
| 24 | **VolatileImage createVolatileImage(int width,int height, ImageCapabilities caps)** Creates a volatile off-screen drawable image, with the given capabilities. |
| 25 | **void deliverEvent(Event e)** Deprecated. As of JDK version 1.1, replaced by dispatchEvent(AWTEvent e). |
| 26 | **void disable()** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean). |
| 27 | **protected void disableEvents(long eventsToDisable)** Disables the events defined by the specified event mask parameter from being delivered to this component. |
| 28 | **void dispatchEvent(AWTEvent e)** Dispatches an event to this component or one of its sub components. |
| 29 | **void doLayout()** Prompts the layout manager to lay out this component. |
| 30 | **void enable()** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean). |

| | |
|---|---|
| 31 | **void enable(boolean b)** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean). |
| 32 | **protected void enableEvents(long eventsToEnable)** Enables the events defined by the specified event mask parameter to be delivered to this component. |
| 33 | **void enableInputMethods(boolean enable)** Enables or disables input method support for this component. |
| 34 | **protected void firePropertyChange(String propertyName, boolean oldValue, boolean newValue)** Support for reporting bound property changes for boolean properties. |
| 35 | **void firePropertyChange(String propertyName, byte oldValue, byte newValue)** Reports a bound property change. |
| 36 | **void firePropertyChange(String propertyName, char oldValue, char newValue)** Reports a bound property change. |
| 37 | **void firePropertyChange(String propertyName, double oldValue, double newValue)** Reports a bound property change. |
| 38 | **void firePropertyChange(String propertyName, float oldValue, float newValue)** Reports a bound property change. |
| 39 | **void firePropertyChange(String propertyName, long oldValue, long newValue)** Reports a bound property change. |
| 40 | **protected void firePropertyChange(String propertyName, Object oldValue, Object newValue)** Support for reporting bound property changes for Object properties. |
| 41 | **void firePropertyChange(String propertyName, short oldValue, short newValue)** Reports a bound property change. |
| 42 | **AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Component. |
| 43 | **float getAlignmentX()** Returns the alignment along the x axis. |
| 44 | **float getAlignmentY()** Returns the alignment along the y axis. |
| 45 | **Color getBackground()** Gets the background color of this component. |
| 46 | **int getBaseline(int width,int height)** Returns the baseline. |
| 47 | **Component.BaselineResizeBehavior getBaselineResizeBehavior()** Returns an enum indicating how the baseline of the component changes as the size changes. |
| 48 | **Rectangle getBounds()** Gets the bounds of this component in the form of a Rectangle object. |

| | |
|---|---|
| 49 | **Rectangle getBounds(Rectangle rv)** Stores the bounds of this component into **return value** rv and return rv. |
| 50 | **ColorModel getColorModel()** Gets the instance of ColorModel used to display the component on the output device. |
| 51 | **Component getComponentAt(int x,int y)** Determines if this component or one of its immediate subcomponents contains the (x, y) location, and if so, returns the containing component. |
| 52 | **Component getComponentAt(Point p)** Returns the component or subcomponent that contains the specified point. |
| 53 | **ComponentListener[] getComponentListeners()** Returns an array of all the component listeners registered on this component. |
| 54 | **ComponentOrientation getComponentOrientation()** Retrieves the language-sensitive orientation that is to be used to order the elements or text within this component. |
| 55 | **Cursor getCursor()** Gets the cursor set in the component. |
| 56 | **DropTarget getDropTarget()** Gets the DropTarget associated with this Component. |
| 57 | **Container getFocusCycleRootAncestor()** Returns the Container which is the focus cycle root of this Component's focus traversal cycle. |
| 58 | **FocusListener[] getFocusListeners()** Returns an array of all the focus listeners registered on this component. |
| 59 | **Set<AWTKeyStroke> getFocusTraversalKeys(int id)** Returns the Set of focus traversal keys for a given traversal operation for this Component. |
| 60 | **boolean getFocusTraversalKeysEnabled()** Returns whether focus traversal keys are enabled for this Component. |
| 61 | **Font getFont()** Gets the font of this component. |
| 62 | **FontMetrics getFontMetrics(Font font)** Gets the font metrics for the specified font. |
| 63 | **Color getForeground()** Gets the foreground color of this component. |
| 64 | **Graphics getGraphics()** Creates a graphics context for this component. |
| 65 | **GraphicsConfiguration getGraphicsConfiguration()** Gets the GraphicsConfiguration associated with this Component. |
| 66 | **int getHeight()** Returns the current height of this component. |
| 67 | **HierarchyBoundsListener[] getHierarchyBoundsListeners()** Returns an array of all the hierarchy bounds listeners registered on this |

| | |
|---|---|
| | component. |
| 68 | **HierarchyListener[] getHierarchyListeners()** Returns an array of all the hierarchy listeners registered on this component. |
| 69 | **boolean getIgnoreRepaint()** |
| 70 | **InputContext getInputContext()** Gets the input context used by this component for handling the communication with input methods when text is entered in this component. |
| 71 | **InputMethodListener[] getInputMethodListeners()** Returns an array of all the input method listeners registered on this component. |
| 72 | **InputMethodRequests getInputMethodRequests()** Gets the input method request handler which supports requests from input methods for this component. |
| 73 | **KeyListener[] getKeyListeners()** Returns an array of all the key listeners registered on this component. |
| 74 | **<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Component. |
| 75 | **Locale getLocale()** Gets the locale of this component. |
| 76 | **Point getLocation()** Gets the location of this component in the form of a point specifying the component's top-left corner. |
| 77 | **Point getLocation(Point rv)** Stores the x,y origin of this component into **return value** rv and return rv. |
| 78 | **Point getLocationOnScreen()** Gets the location of this component in the form of a point specifying the component's top-left corner in the screen's coordinate space. |
| 79 | **Dimension getMaximumSize()** Gets the maximum size of this component. |
| 80 | **Dimension getMinimumSize()** Gets the mininimum size of this component. |
| 81 | **MouseListener[] getMouseListeners()** Returns an array of all the mouse listeners registered on this component. |
| 82 | **MouseMotionListener[] getMouseMotionListeners()** Returns an array of all the mouse motion listeners registered on this component. |
| 83 | **Point getMousePosition()** Returns the position of the mouse pointer in this Component's coordinate space if the Component is directly under the mouse pointer, otherwise returns null. |
| | **MouseWheelListener[] getMouseWheelListeners()** Returns an array |

| | |
|---|---|
| | of all the mouse wheel listeners registered on this component. |
| 85 | **String getName()** Gets the name of the component. |
| 86 | **Container getParent()** Gets the parent of this component. |
| 87 | **java.awt.peer.ComponentPeer getPeer() Deprecated. As of JDK version 1.1, programs should not directly manipulate peers; replaced by boolean isDisplayable().** |
| 88 | **Dimension getPreferredSize()** Gets the preferred size of this component. |
| 89 | **PropertyChangeListener[] getPropertyChangeListeners()** Returns an array of all the property change listeners registered on this component. |
| 90 | **PropertyChangeListener[] getPropertyChangeListeners(String propertyName)** Returns an array of all the listeners which have been associated with the named property. |
| 91 | **Dimension getSize()** Returns the size of this component in the form of a Dimension object. |
| 92 | **Dimension getSize(Dimension rv)Stores the width/height of this component into** return value **rv and return rv.** |
| 93 | **Toolkit getToolkit()** Gets the toolkit of this component. |
| 94 | **Object getTreeLock()** Gets this component's locking object (the object that owns the thread sychronization monitor) for AWT component-tree and layout operations. |
| 95 | **int getWidth()** Returns the current width of this component. |
| 96 | **int getX()** Returns the current x coordinate of the components origin. |
| 97 | **int getY()** Returns the current y coordinate of the components origin. |
| 98 | **boolean gotFocus(Event evt, Object what)** Deprecated. As of JDK version 1.1, replaced by processFocusEvent(FocusEvent) . |
| 99 | **boolean handleEvent(Event evt)** Deprecated. As of JDK version 1.1 replaced by processEvent(AWTEvent). |
| 100 | **boolean hasFocus()** Returns true if this Component is the focus owner. |
| 101 | **void hide()** Deprecated. As of JDK version 1.1, replaced by setVisible(boolean). |
| 102 | **boolean imageUpdate(Image img,int infoflags,int x,int y,int w,int h)** Repaints the component when the image has changed. |
| | **boolean inside(int x,int y)** Deprecated. As of JDK version 1.1, |

| | |
|---|---|
| 103 | **boolean inside(int x,int y)** Deprecated. As of JDK version 1.1, replaced by contains(int, int). |
| 104 | **void invalidate()** Invalidates this component. |
| 105 | **boolean isBackgroundSet()** Returns whether the background color has been explicitly set for this Component. |
| 106 | **boolean isCursorSet()** Returns whether the cursor has been explicitly set for this Component. |
| 107 | **boolean isDisplayable()** Determines whether this component is displayable. |
| 108 | **boolean isDoubleBuffered()** Returns true if this component is painted to an offscreen image (**buffer**) that's copied to the screen later. |
| 109 | **boolean isEnabled()** Determines whether this component is enabled. |
| 110 | **boolean isFocusable()** Returns whether this Component can be focused. |
| 111 | **boolean isFocusCycleRoot(Container container)** Returns whether the specified Container is the focus cycle root of this Component's focus traversal cycle. |
| 112 | **boolean isFocusOwner()** Returns true if this Component is the focus owner. |
| 113 | **boolean isFocusTraversable()** Deprecated. As of 1.4, replaced by isFocusable(). |
| 114 | **boolean isFontSet()** Returns whether the font has been explicitly set for this Component. |
| 115 | **boolean isForegroundSet()** Returns whether the foreground color has been explicitly set for this Component. |
| 116 | **boolean isLightweight()** A lightweight component doesn't have a native toolkit peer. |
| 117 | **boolean isMaximumSizeSet()** Returns true if the maximum size has been set to a non-null value otherwise returns false. |
| 118 | **boolean isMinimumSizeSet()** Returns whether or not setMinimumSize has been invoked with a non-null value. |
| 119 | **boolean isOpaque()** Returns true if this component is completely opaque, returns false by default. |
| 120 | **boolean isPreferredSizeSet()** Returns true if the preferred size has been set to a non-null value otherwise returns false. |
| 121 | **boolean isShowing()** Determines whether this component is showing on screen. |

| | |
|---|---|
| 122 | **boolean isValid()** Determines whether this component is valid. |
| 123 | **boolean isVisible()** Determines whether this component should be visible when its parent is visible. |
| 124 | **boolean keyDown(Event evt,int key)** Deprecated. As of JDK version 1.1, replaced by processKeyEvent(KeyEvent). |
| 125 | **boolean keyUp(Event evt,int key)** Deprecated. As of JDK version 1.1, replaced by processKeyEvent(KeyEvent). |
| 126 | **void layout()** Deprecated. As of JDK version 1.1, replaced by doLayout(). |
| 127 | **void list()** Prints a listing of this component to the standard system output stream System.out. |
| 128 | **void list(PrintStream out)** Prints a listing of this component to the specified output stream. |
| 129 | **void list(PrintStream out,int indent)** Prints out a list, starting at the specified indentation, to the specified print stream. |
| 130 | **void list(PrintWriter out)** Prints a listing to the specified print writer. |
| 131 | **void list(PrintWriter out,int indent)** Prints out a list, starting at the specified indentation, to the specified print writer. |
| 132 | **Component locate(int x,int y)** Deprecated. As of JDK version 1.1, replaced by getComponentAt(int, int). |
| 133 | **Point location()** Deprecated. As of JDK version 1.1, replaced by getLocation(). |
| 134 | **boolean lostFocus(Event evt, Object what)** Deprecated. As of JDK version 1.1, replaced by processFocusEvent(FocusEvent). |
| 135 | **boolean mouseDown(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent). |
| 136 | **boolean mouseDrag(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseMotionEvent(MouseEvent). |
| 137 | **boolean mouseEnter(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent). |
| 138 | **boolean mouseExit(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent).. |
| 139 | **boolean mouseMove(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseMotionEvent(MouseEvent).. |
| 140 | **boolean mouseUp(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent). |

| | |
|---|---|
| 141 | **void move(int x,int y)** Deprecated. As of JDK version 1.1, replaced by setLocation(int, int). |
| 142 | **void nextFocus()** Deprecated. As of JDK version 1.1, replaced by transferFocus(). |
| 143 | **void paint(Graphics g)** Paints this component. |
| 144 | **void paintAll(Graphics g)** Paints this component and all of its subcomponents. |
| 145 | **boolean postEvent(Event e)** Deprecated. As of JDK version 1.1, replaced by dispatchEvent(AWTEvent). |
| 146 | **boolean prepareImage(Image image,int width,int height, ImageObserver observer)** Prepares an image for rendering on this component at the specified width and height. |
| 147 | **void print(Graphics g)** Prints this component. |
| 148 | **void printAll(Graphics g)** Prints this component and all of its subcomponents. |
| 149 | **protectedvoid processComponentEvent(ComponentEvent e)** Processes component events occurring on this component by dispatching them to any registered ComponentListener objects. |
| 150 | **protected void processEvent(AWTEvent e)** Processes events occurring on this component. |
| 151 | **protected void processFocusEvent(FocusEvent e)** Processes focus events occurring on this component by dispatching them to any registered FocusListener objects. |
| 152 | **protected void processHierarchyBoundsEvent(HierarchyEvent e)** Processes hierarchy bounds events occurring on this component by dispatching them to any registered HierarchyBoundsListener objects. |
| 153 | **protected void processHierarchyEvent(HierarchyEvent e)** Processes hierarchy events occurring on this component by dispatching them to any registered HierarchyListener objects. |
| 154 | **protectedvoid processInputMethodEvent(InputMethodEvent e)** Processes input method events occurring on this component by dispatching them to any registered InputMethodListener objects. |
| 155 | **protected void processKeyEvent(KeyEvent e)** Processes key events occurring on this component by dispatching them to any registered KeyListener objects. |
| 156 | **protected void processMouseEvent(MouseEvent e)** Processes mouse events occurring on this component by dispatching them to any registered MouseListener objects. |

| | |
|---|---|
| 157 | **protected void processMouseMotionEvent(MouseEvent e)** Processes mouse motion events occurring on this component by dispatching them to any registered MouseMotionListener objects. |
| 158 | **protected void processMouseWheelEvent(MouseWheelEvent e)** Processes mouse wheel events occurring on this component by dispatching them to any registered MouseWheelListener objects. |
| 159 | **void remove(MenuComponent popup)** Removes the specified popup menu from the component. |
| 160 | **void removeComponentListener(ComponentListener l)** Removes the specified component listener so that it no longer receives component events from this component. |
| 161 | **void removeFocusListener(FocusListener l)** Removes the specified focus listener so that it no longer receives focus events from this component. |
| 162 | **void removeHierarchyBoundsListener(HierarchyBoundsListener l)** Removes the specified hierarchy bounds listener so that it no longer receives hierarchy bounds events from this component. |
| 163 | **void removeHierarchyListener(HierarchyListener l)** Removes the specified hierarchy listener so that it no longer receives hierarchy changed events from this component. |
| 164 | **void removeInputMethodListener(InputMethodListener l)** Removes the specified input method listener so that it no longer receives input method events from this component. |
| 165 | **void removeKeyListener(KeyListener l)** Removes the specified key listener so that it no longer receives key events from this component. |
| 166 | **void removeMouseListener(MouseListener l)** Removes the specified mouse listener so that it no longer receives mouse events from this component. |
| 167 | **void removeMouseMotionListener(MouseMotionListener l)** Removes the specified mouse motion listener so that it no longer receives mouse motion events from this component. |
| 168 | **void removeMouseWheelListener(MouseWheelListener l)** Removes the specified mouse wheel listener so that it no longer receives mouse wheel events from this component. |
| 169 | **void removeNotify()** Makes this Component undisplayable by destroying it native screen resource. |
| 170 | **void removePropertyChangeListener(PropertyChangeListener listener)** Removes a PropertyChangeListener from the listener list. |
| | **void removePropertyChangeListener(String propertyName,** |

| | |
|---|---|
| 171 | **PropertyChangeListener listener)** Removes a PropertyChangeListener from the listener list for a specific property. |
| 172 | **void repaint()** Repaints this component. |
| 173 | **void repaint(int x,int y,int width,int height)** Repaints the specified rectangle of this component. |
| 174 | **void repaint(long tm)** Repaints the component. |
| 175 | **void repaint(long tm,int x,int y,int width,int height)** Repaints the specified rectangle of this component within tm milliseconds. |
| 176 | **void requestFocus()** Requests that this Component get the input focus, and that this Component's top-level ancestor become the focused Window. |
| 177 | **protected boolean requestFocus(boolean temporary)** Requests that this Component get the input focus, and that this Component's top-level ancestor become the focused Window. |
| 178 | **boolean requestFocusInWindow()** Requests that this Component get the input focus, if this Component's top-level ancestor is already the focused Window. |
| 179 | **protected boolean requestFocusInWindow(boolean temporary)** Requests that this Component get the input focus, if this Component's top-level ancestor is already the focused Window. |
| 180 | **void reshape(int x,int y,int width,int height)** Deprecated. As of JDK version 1.1, replaced by setBounds(int, int, int, int). |
| 181 | **void resize(Dimension d)** Deprecated. As of JDK version 1.1, replaced by setSize(Dimension). |
| 182 | **void resize(int width,int height)** Deprecated. As of JDK version 1.1, replaced by setSize(int, int). |
| 183 | **void setBackground(Color c)** Sets the background color of this component. |
| 184 | **void setBounds(int x,int y,int width,int height)** Moves and resizes this component. |
| 185 | **void setBounds(Rectangle r)** Moves and resizes this component to conform to the new bounding rectangle r. |
| 186 | **void setComponentOrientation(ComponentOrientation o)** Sets the language-sensitive orientation that is to be used to order the elements or text within this component. |
| 187 | **void setCursor(Cursor cursor)** Sets the cursor image to the specified cursor. |

| | |
|---|---|
| 188 | **void setDropTarget(DropTarget dt)** Associate a DropTarget with this component. |
| 189 | **void setEnabled(boolean b)** Enables or disables this component, depending on the value of the parameter b. |
| 190 | **void setFocusable(boolean focusable)** Sets the focusable state of this Component to the specified value. |
| 191 | **void setFocusTraversalKeys(int id, Set<? extends AWTKeyStroke> keystrokes)** Sets the focus traversal keys for a given traversal operation for this Component. |
| 192 | **void setFocusTraversalKeysEnabled(boolean focusTraversalKeysEnabled)** Sets whether focus traversal keys are enabled for this Component. |
| 193 | **void setFont(Font f)** Sets the font of this component. |
| 194 | **void setForeground(Color c)** Sets the foreground color of this component. |
| 195 | **void setIgnoreRepaint(boolean ignoreRepaint)** Sets whether or not paint messages received from the operating system should be ignored. |
| 196 | **void setLocale(Locale l)** Sets the locale of this component. |
| 197 | **void setLocation(int x,int y)** Moves this component to a new location. |
| 198 | **void setLocation(Point p)** Moves this component to a new location. |
| 199 | **void setMaximumSize(Dimension maximumSize)** Sets the maximum size of this component to a constant value. |
| 200 | **void setMinimumSize(Dimension minimumSize)** Sets the minimum size of this component to a constant value. |
| 201 | **void setName(String name)** Sets the name of the component to the specified string. |
| 202 | **void setPreferredSize(Dimension preferredSize)** Sets the preferred size of this component to a constant value. |
| 203 | **void setSize(Dimension d)** Resizes this component so that it has width d.width and height d.height. |
| 204 | **void setSize(int width,int height)** Resizes this component so that it has width width and height height. |
| 205 | **void setVisible(boolean b)** Shows or hides this component depending on the value of parameter b. |
| 206 | **void show()** Deprecated. As of JDK version 1.1, replaced by setVisible(boolean). |
| | |

| | |
|---|---|
| 207 | **void show(boolean b)** Deprecated. As of JDK version 1.1, replaced by setVisible(boolean). |
| 208 | **Dimension size()** Deprecated. As of JDK version 1.1, replaced by getSize(). |
| 209 | **String toString()** Returns a string representation of this component and its values. |
| 210 | **void transferFocus()** Transfers the focus to the next component, as though this Component were the focus owner. |
| 211 | **void transferFocusBackward()** Transfers the focus to the previous component, as though this Component were the focus owner. |
| 212 | **void transferFocusUpCycle()** Transfers the focus up one focus traversal cycle. |
| 213 | **void update(Graphics g)** Updates this component. |
| 214 | **void validate()** Ensures that this component has a valid layout. |
| 215 | **Rectangle bounds()** Deprecated. As of JDK version 1.1, replaced by getBounds(). |
| 216 | **protected AWTEvent coalesceEvents(AWTEvent existingEvent, AWTEvent newEvent)** Potentially coalesce an event being posted with an existing event. |
| 217 | **protected String paramString()** Returns a string representing the state of this component. |
| 218 | **protected void firePropertyChange(String propertyName,int oldValue,int newValue)** Support for reporting bound property changes for integer properties. |
| 219 | **Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(). |
| 220 | **boolean prepareImage(Image image, ImageObserver observer)** Prepares an image for rendering on this component. |
| 221 | **Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(). |

## 继承的方法

这个类继承的方法从以下类：

- java.lang.Object

# AWT Label类 - AWT

## 介绍

标签是一种被动的控制，因为它不会由用户访问产生任何事件。 Label控件是一个对象标签。一个标签显示一行只读文本。然而，文本可以由应用程序改变，但不能由最终用户以任何方式改变。

## 类的声明

以下是声明的java.awt.Label类：

```
public class Label
   extends Component
      implements Accessible
```

## 字段域

以下是java.awt.Component 类的字段：

- static int CENTER -- 表示标签应为中心对齐。
- static int LEFT -- 表示应左对齐标签。
- static int RIGHT -- 表示应右对齐标签。

## 类的构造函数

| S.N. | 构造函数与说明 |
|---|---|
| 1 | **Label()** Constructs an empty label. |
| 2 | **Label(String text)** Constructs a new label with the specified string of text, left justified. |
| 3 | **Label(String text, int alignment)** Constructs a new label that presents the specified string of text with the specified alignment. |

## 类方法

| S.N. | 方法和说明 |
|---|---|
| 1 | **void addNotify()** Creates the peer for this label. |
| 2 | **AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Label. |
| 3 | **int getAlignment()** Gets the current alignment of this label. |
| 4 | **String getText()** Gets the text of this label. |
| 5 | **protected String paramString()** Returns a string representing the state of this Label. |
| 6 | **void setAlignment(int alignment)** Sets the alignment for this label to the specified alignment. |
| 7 | **void setText(String text)** Sets the text for this label to the specified text. |

## 继承的方法

这个类继承的方法从以下类：

- java.awt.Component
- java.lang.Object

## 标签示例

说在您选择使用编辑器创建以下java程序 D:/ > AWT > com > yiibai.com > gui >

AwtControlDemo

```
package com.yiibai.gui;

import java.awt.*;
import java.awt.event.*;

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }
```

```
    public static void main(String[] args){
       AwtControlDemo  awtControlDemo = new AwtControlDemo();
       awtControlDemo.showLabelDemo();
    }

    private void prepareGUI(){
       mainFrame = new Frame("Java AWT Examples");
       mainFrame.setSize(400,400);
       mainFrame.setLayout(new GridLayout(3, 1));
       mainFrame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
          }
       });
       headerLabel = new Label();
       headerLabel.setAlignment(Label.CENTER);
       statusLabel = new Label();
       statusLabel.setAlignment(Label.CENTER);
       statusLabel.setSize(350,100);

       controlPanel = new Panel();
       controlPanel.setLayout(new FlowLayout());

       mainFrame.add(headerLabel);
       mainFrame.add(controlPanel);
       mainFrame.add(statusLabel);
       mainFrame.setVisible(true);
    }

    private void showLabelDemo(){
       headerLabel.setText("Control in action: Label");

       Label label = new Label();
       label.setText("Welcome to TutorialsPoint AWT Tutorial.");
       label.setAlignment(Label.CENTER);
       label.setBackground(Color.GRAY);
       label.setForeground(Color.WHITE);
       controlPanel.add(label);

       mainFrame.setVisible(true);
    }
}
```

编译程序，使用命令提示符。到 **D:/ > AWT** 然后键入以下命令。

```
D:AWT>javac comyiibaiguiAwtControlDemo.java
```

如果没有错误出现，这意味着编译成功。使用下面的命令来运行程序。

```
D:AWT>java com.yiibai.gui.AwtControlDemo
```

验证下面的输出



# AWT Button类 - AWT

## 介绍

按钮是一个控制组件，按下时有一个标签，并生成一个事件。当按钮被按下和释放，AWT发送ActionEvent的一个实例的按钮，通过调用按钮上的processEvent。按钮的processEvent方法接收所有事件的按钮，它通过一个动作事件以及调用其自身的processActionEvent方法。后一种方法通过操作事件的动作的已注册侦听此按钮所产生的动作事件。

如果一个应用程序需要一个按钮被按下和释放的基础上执行一些动作，但要实现ActionListener并注册新的侦听器接收事件从这个按钮，调用按钮的addActionListener方法。应用程序可以利用按钮的动作命令的消息传递协议。

## 类的声明

以下是声明的 java.awt.Button类 :

```
public class Button
   extends Component
      implements Accessible
```

## 类的构造函数

| S.N. | 构造函数与说明 |
| --- | --- |
| 1 | **Button()** Constructs a button with an empty string for its label. |
| 2 | **Button(String text)** Constructs a new button with specified label. |

## 类方法

| S.N. | 方法和说明 |
| --- | --- |
| 1 | **void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this button. |
| 2 | **void addNotify()** Creates the peer of the button. |
| 3 | **AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Button. |
| 4 | **String getActionCommand()** Returns the command name of the action event fired by this button. |
| 5 | **ActionListener[] getActionListeners()** Returns an array of all the action listeners registered on this button. |
| 6 | **String getLabel()** Gets the label of this button. |
| 7 | **<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Button. |
| 8 | **protected String paramString()** Returns a string representing the state of this Button. |
| 9 | **protected void processActionEvent(ActionEvent e)** Processes action events occurring on this button by dispatching them to any registered ActionListener objects. |
| 10 | **protected void processEvent(AWTEvent e)** Processes events on this button. |
| 11 | **void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this button. |
| 12 | **void setActionCommand(String command)** Sets the command name for the action event fired by this button. |
| 13 | **void setLabel(String label)** Sets the button's label to be the specified string. |

## 继承的方法

这个类继承的方法从以下类：

- java.awt.Component
- java.lang.Object

## 按钮实例

选择使用编辑器创建以下java程序 D:/ > AWT > com > yiibai.com > gui >

AwtControlDemo.java

```
package com.yiibai.gui;

import java.awt.*;
import java.awt.event.*;

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showButtonDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showButtonDemo(){
      headerLabel.setText("Control in action: Button");
```

```
        Button okButton = new Button("OK");
        Button submitButton = new Button("Submit");
        Button cancelButton = new Button("Cancel");

        okButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              statusLabel.setText("Ok Button clicked.");
           }
        });

        submitButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              statusLabel.setText("Submit Button clicked.");
           }
        });

        cancelButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              statusLabel.setText("Cancel Button clicked.");
           }
        });

        controlPanel.add(okButton);
        controlPanel.add(submitButton);
        controlPanel.add(cancelButton);

        mainFrame.setVisible(true);
    }
}
```

编译程序，使用命令提示符。到 **D:/ > AWT** 然后键入以下命令。

```
D:AWT>javac comyiibaiguiAwtControlDemo.java
```

如果没有错误出现，这意味着编译成功。使用下面的命令来运行程序。

```
D:AWT>java com.yiibai.gui.AwtControlDemo
```

验证下面的输出


Java AWT Examples
Control in action: Button
OK
Submit
Cancel
Ok Button clicked.
Yiibai.com

# AWT CheckBox类 - AWT

## 介绍

CheckBox控件是用来开启选项（true）或关闭（false）。每个复选框标签代表的复选框做什么么。通过点击它可以改变一个复选框的状态。

## 类的声明

以下是声明的java.awt.Checkbox类：

```
public class Checkbox
   extends Component
      implements ItemSelectable,Accessible
```

## 类的构造函数

| S.N. | 构造函数与说明 |
| --- | --- |
| 1 | **Checkbox()** Creates a check box with an empty string for its label. |
| 2 | **Checkbox(String label)** Creates a check box with the specified label. |
| 3 | **Checkbox(String label, boolean state)** Creates a check box with the specified label and sets the specified state. |
| 4 | **Checkbox(String label, boolean state, CheckboxGroup group)** Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group. |
| 5 | **Checkbox(String label, CheckboxGroup group, boolean state)** Creates a check box with the specified label, in the specified check box group, and set to the specified state. |

## 类方法

| S.N. | 方法和说明 |
|---|---|
| 1 | **void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this check box. |
| 2 | **void addNotify()** Creates the peer of the Checkbox. |
| 3 | **AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Checkbox. |
| 4 | **CheckboxGroup getCheckboxGroup()** Determines this check box's group. |
| 5 | **ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this checkbox. |
| 6 | **String getLabel()** Gets the label of this check box. |
| 7 | **<T extends EventListener>T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Checkbox. |
| 8 | **Object[] getSelectedObjects()** Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected. |
| 9 | **boolean getState()** Determines whether this check box is in the **on** or **off** state. |
| 10 | **protected String paramString()** Returns a string representing the state of this Checkbox. |
| 11 | **protected void processEvent(AWTEvent e)** Processes events on this check box. |
| 12 | **protected void processItemEvent(ItemEvent e)** Processes item events occurring on this check box by dispatching them to any registered ItemListener objects. |
| 13 | **void removeItemListener(ItemListener l)** Removes the specified item listener so that the item listener no longer receives item events from this check box. |
| 14 | **void setCheckboxGroup(CheckboxGroup g)** Sets this check box's group to the specified check box group. |
| 15 | **void setLabel(String label)** Sets this check box's label to be the string argument. |
| 16 | **void setState(boolean state)** Sets the state of this check box to the specified state. |

## 继承的方法

这个类继承的方法从以下类：

- java.awt.Component
- java.lang.Object

# CheckBox的示例

说在您选择使用的编辑器创建以下java程序 D:/ > AWT > com > yiibai.com > gui > AwtControlDemo.java

```
package com.yiibai.gui;

import java.awt.*;
import java.awt.event.*;

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showCheckBoxDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());
```

```
        mainFrame.add(headerLabel);
        mainFrame.add(controlPanel);
        mainFrame.add(statusLabel);
        mainFrame.setVisible(true);
    }

    private void showCheckBoxDemo(){

        headerLabel.setText("Control in action: CheckBox");

        Checkbox chkApple = new Checkbox("Apple");
        Checkbox chkMango = new Checkbox("Mango");
        Checkbox chkPeer = new Checkbox("Peer");

        chkApple.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                statusLabel.setText("Apple Checkbox: "
                + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        chkMango.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                statusLabel.setText("Mango Checkbox: "
                + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        chkPeer.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                statusLabel.setText("Peer Checkbox: "
                + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        controlPanel.add(chkApple);
        controlPanel.add(chkMango);
        controlPanel.add(chkPeer);

        mainFrame.setVisible(true);
    }
}
```

编译程序，使用命令提示符。到 **D:/ > AWT** 然后键入以下命令。

```
D:AWT>javac comyiibaiguiAwtControlDemo.java
```

如果没有错误出现，这意味着编译成功。使用下面的命令来运行程序。

```
D:AWT>java com.yiibai.gui.AwtControlDemo
```

验证下面的输出



# AWT List类 - AWT

## 介绍

List是一个文本项列表。该列表可以被配置给该用户可以选择一个或多个项目。

## 类的声明

以下是的声明类java.awt.List：

```
public class List
   extends Component
      implements ItemSelectable, Accessible
```

## 类的构造函数

| S.N. | 构造函数与说明 |
| --- | --- |
| 1 | **List()** Creates a new scrolling list. |
| 2 | **List(int rows)** Creates a new scrolling list initialized with the specified number of visible lines. |
| 3 | **List(int rows, boolean multipleMode)** Creates a new scrolling list initialized to display the specified number of rows. |

## 类方法

T[] getListeners(Class listenerType) 目前注册的所有对象作为FooListeners名单后返回一个数组。

| S.N. | Method & Description |
| --- | --- |
| 1 | **void add(String item)** Adds the specified item to the end of scrolling list. |
| 2 | **void add(String item, int index)** Adds the specified item to the the scrolling list at the position indicated by the index. |
| 3 | **void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this list. |
| | |

| | |
|---|---|
| 4 | **void addItem(String item)** Deprecated. replaced by add(String). |
| 5 | **void addItem(String item, int index)** Deprecated. replaced by add(String, int). |
| 6 | **void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this list. |
| 7 | **void addNotify()** Creates the peer for the list. |
| 8 | **boolean allowsMultipleSelections()** Deprecated. As of JDK version 1.1, replaced by isMultipleMode(). |
| 9 | **void clear()** Deprecated. As of JDK version 1.1, replaced by removeAll(). |
| 10 | **int countItems()** Deprecated. As of JDK version 1.1, replaced by getItemCount(). |
| 11 | **void delItem(int position)** Deprecated. replaced by remove(String) and remove(int). |
| 12 | **void delItems(int start, int end)** Deprecated. As of JDK version 1.1, Not for public use in the future. This method is expected to be retained only as a package private method. |
| 13 | **void deselect(int index)** Deselects the item at the specified index. |
| 14 | **AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this List. |
| 15 | **ActionListener[] getActionListeners()** Returns an array of all the action listeners registered on this list. |
| 16 | **String getItem(int index)** Gets the item associated with the specified index. |
| 17 | **int getItemCount()** Gets the number of items in the list. |
| 18 | **ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this list. |
| 19 | **String[] getItems()** Gets the items in the list. |
| 20 | **Dimension getMinimumSize()** Determines the minimum size of this scrolling list. |
| 21 | **Dimension getMinimumSize(int rows)** Gets the minumum dimensions for a list with the specified number of rows. |
| 22 | **Dimension getPreferredSize()** Gets the preferred size of this scrolling list. |
| 23 | **Dimension getPreferredSize(int rows)** Gets the preferred dimensions for a list with the specified number of rows. |

| | |
|---|---|
| 24 | **int getRows()** Gets the number of visible lines in this list. |
| 25 | **int getSelectedIndex()** Gets the index of the selected item on the list, |
| 26 | **int[] getSelectedIndexes()** Gets the selected indexes on the list. |
| 27 | **String getSelectedItem()** Gets the selected item on this scrolling list. |
| 28 | **String[] getSelectedItems()** Gets the selected items on this scrolling list. |
| 29 | **Object[] getSelectedObjects()** Gets the selected items on this scrolling list in an array of Objects. |
| 30 | **int getVisibleIndex()** Gets the index of the item that was last made visible by the method makeVisible. |
| 31 | **boolean isIndexSelected(int index)** Determines if the specified item in this scrolling list is selected. |
| 32 | **boolean isMultipleMode()** Determines whether this list allows multiple selections. |
| 33 | **boolean isSelected(int index)** Deprecated. As of JDK version 1.1, replaced by isIndexSelected(int). |
| 34 | **void makeVisible(int index)** Makes the item at the specified index visible. |
| 35 | **Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(). |
| 36 | **Dimension minimumSize(int rows)** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(int). |
| 37 | **protected String paramString()** Returns the parameter string representing the state of this scrolling list. |
| 38 | **Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(). |
| 39 | **Dimension preferredSize(int rows)** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(int). |
| 40 | **protected void processActionEvent(ActionEvent e)** Processes action events occurring on this component by dispatching them to any registered ActionListener objects. |
| 41 | **protected void processEvent(AWTEvent e)** Processes events on this scrolling list. |
| 42 | **protected void processItemEvent(ItemEvent e)** Processes item events occurring on this list by dispatching them to any registered ItemListener objects. |

| 43 | **void remove(int position)** Removes the item at the specified position from this scrolling list. |
|---|---|
| 44 | **void remove(String item)** Removes the first occurrence of an item from the list. |
| 45 | **void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this list. |
| 46 | **void removeAll()** Removes all items from this list. |
| 47 | **void removeItemListener(ItemListener l)** Removes the specified item listener so that it no longer receives item events from this list. |
| 48 | **void removeNotify()** Removes the peer for this list. |
| 49 | **void replaceItem(String newValue, int index)** Replaces the item at the specified index in the scrolling list with the new string. |
| 50 | **void select(int index)** Selects the item at the specified index in the scrolling list. |
| 51 | **void setMultipleMode(boolean b)** Sets the flag that determines whether this list allows multiple selections. |
| 52 | **void setMultipleSelections(boolean b)** Deprecated. As of JDK version 1.1, replaced by setMultipleMode(boolean). |

## 继承的方法

这个类从以下类继承的方法：

- java.awt.Component
- java.lang.Object

# List 示例

选择使用任何编辑器创建以下java程序 D:/ > AWT > com > yiibai.com > gui >

AwtControlDemo

```
package com.yiibai.gui;

import java.awt.*;
import java.awt.event.*;

public class AwtControlDemo {
```

```
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showListDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showListDemo(){

      headerLabel.setText("Control in action: List");
      final List fruitList = new List(4,false);

      fruitList.add("Apple");
      fruitList.add("Grapes");
      fruitList.add("Mango");
      fruitList.add("Peer");

      final List vegetableList = new List(4,true);

      vegetableList.add("Lady Finger");
      vegetableList.add("Onion");
      vegetableList.add("Potato");
```

```
        vegetableList.add("Tomato");

        Button showButton = new Button("Show");

        showButton.addActionListener(new ActionListener() {

           public void actionPerformed(ActionEvent e) {
              String data = "Fruits Selected: "
                 + fruitList.getItem(fruitList.getSelectedIndex());
              data += ", Vegetables selected: ";
              for(String vegetable:vegetableList.getSelectedItems()){
                 data += vegetable + " ";
              }
              statusLabel.setText(data);
           }
        });

        controlPanel.add(fruitList);
        controlPanel.add(vegetableList);
        controlPanel.add(showButton);

        mainFrame.setVisible(true);
     }
  }
```

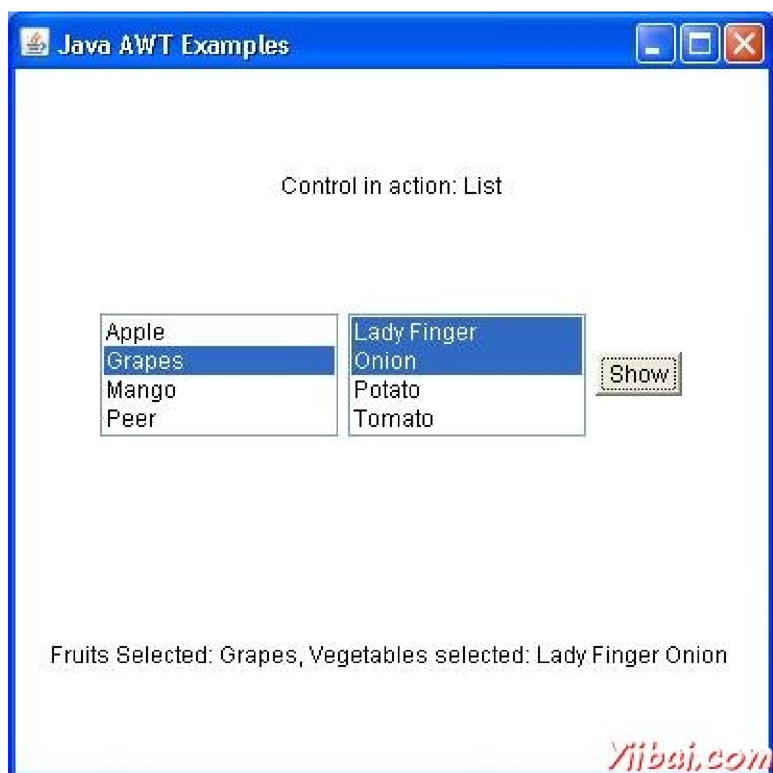
编译程序，使用命令提示符。到 D:/ > AWT 然后键入以下命令。

```
D:AWT>javac comyiibai.comguiAwtControlDemo.java
```

如果没有错误出现，这意味着编译成功。使用下面的命令来运行程序。

```
D:AWT>java com.yiibai.gui.AwtControlDemo
```

验证下面的输出


Java AWT Examples
Control in action: List
Apple
Grapes
Mango
Peer
Lady Finger
Onion
Potato
Tomato
Show
Fruits Selected: Grapes, Vegetables selected: Lady Finger Onion
Yiibai.com

# AWT Choice类 - AWT

## 介绍

选择用于控制显示弹出菜单选择。所选选项将显示在顶部的菜单。

## 类的声明

以下是声明的java.awt.Choice类：

```
public class Choice
   extends Component
      implements ItemSelectable, Accessible
```

## 类的构造函数

| S.N. | 构造函数& 描述 |
|---|---|
| 1 | **Choice() ()** Creates a new choice menu. |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **void add(String item)** Adds an item to this Choice menu. |
| 2 | **void addItem(String item)** Obsolete as of Java 2 platform v1.1. |
| 3 | **void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this Choice menu. |
| 4 | **void addNotify()** Creates the Choice's peer. |
| 5 | **int countItems()** Deprecated. As of JDK version 1.1, replaced by getItemCount(). |
| 6 | **AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Choice. |
| 7 | **String getItem(int index)** Gets the string at the specified index in this Choice menu. |
| 8 | **int getItemCount()** Returns the number of items in this Choice menu. |

| | |
|---|---|
| 9 | **ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this choice. |
| 10 | **<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Choice. |
| 11 | **int getSelectedIndex()** Returns the index of the currently selected item. |
| 12 | **String getSelectedItem()** Gets a representation of the current choice as a string. |
| 13 | **Object[] getSelectedObjects()** Returns an array (length 1) containing the currently selected item. |
| 14 | **void insert(String item, int index)** Inserts the item into this choice at the specified position. |
| 15 | **protected String paramString()** Returns a string representing the state of this Choice menu. |
| 16 | **protected void processEvent(AWTEvent e)** Processes events on this choice. |
| 17 | **protected void processItemEvent(ItemEvent e)** Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects. |
| 18 | **void remove(int position)** Removes an item from the choice menu at the specified position. |
| 19 | **void remove(String item)** Removes the first occurrence of item from the Choice menu. |
| 20 | **void removeAll()** Removes all items from the choice menu. |
| 21 | **void removeItemListener(ItemListener l)** Removes the specified item listener so that it no longer receives item events from this Choice menu. |
| 22 | **void select(int pos)** Sets the selected item in this Choice menu to be the item at the specified position. |
| 23 | **void select(String str)** Sets the selected item in this Choice menu to be the item whose name is equal to the specified string. |

## 继承的方法

这个类继承的方法从以下类：

- java.awt.Component

- java.lang.Object

## Choice 实例

选择使用任何编辑器创建以下java程序 D:/ > AWT > com > yiibai > gui >

AwtControlDemo.java

```
package com.yiibai.gui;

import java.awt.*;
import java.awt.event.*;

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showChoiceDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
```

```
    }

    private void showChoiceDemo(){

       headerLabel.setText("Control in action: Choice");
       final Choice fruitChoice = new Choice();

       fruitChoice.add("Apple");
       fruitChoice.add("Grapes");
       fruitChoice.add("Mango");
       fruitChoice.add("Peer");

       Button showButton = new Button("Show");

       showButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
             String data = "Fruit Selected: "
             + fruitChoice.getItem(fruitChoice.getSelectedIndex());
             statusLabel.setText(data);
          }
       });

       controlPanel.add(fruitChoice);
       controlPanel.add(showButton);

       mainFrame.setVisible(true);
    }
 }
```

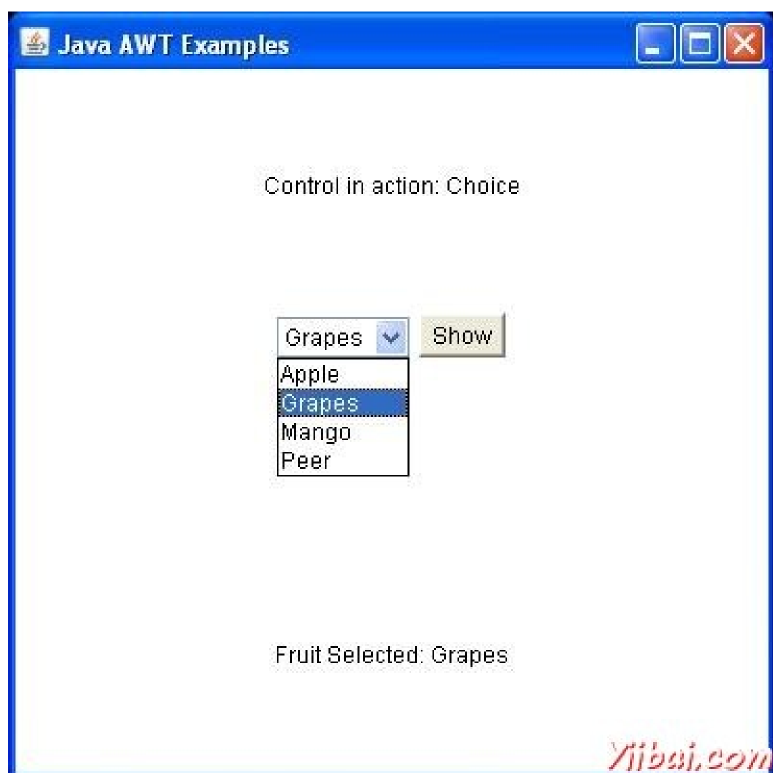
编译程序，使用命令提示符。到 **D:/ > AWT** 然后键入以下命令。

```
D:AWT>javac comyiibaiguiAwtControlDemo.java
```

如果没有错误出现，这意味着编译成功。使用下面的命令来运行程序。

```
D:AWT>java com.yiibai.gui.AwtControlDemo
```

验证下面的输出


Java AWT Examples
Control in action: Choice
Grapes
Show
Apple
Grapes
Mango
Peer
Fruit Selected: Grapes
Yiibai.com

# AWT事件处理 - AWT

## 事件是什么?

一个对象的状态变化被称为事件，即事件描述源状态的变化。事件产生的结果与用户交互的图形用户界面组件。例如，点击一个按钮，移动鼠标，通过键盘输入一个字符，从列表中选择一个项目，滚动页面的活动，使一个事件的发生。

## 事件的类型

事件可以被大致分为两类：

- 前台事件 - 这些事件需要用户直接互动。在图形用户界面中的图形组件交互的人产生的后果。例如，点击一个按钮，移动鼠标，通过键盘输入一个字符，从列表中选择一个项目，滚动页面等
- 后台事件 - 这些事件，需要最终用户的交互是已知的作为后台的事件。操作系统的中断，硬件或软件故障，定时器到期时，操作完成的后台事件的例子。

## 事件处理是什么?

事件处理机制，控制的事件，并决定如果一个事件发生时，会发生什么。这种机制被称为事件处理程序，在事件发生时执行的代码。 Java使用代理事件模型来处理事件。该模型定义了标准的机制来生成和处理事件。让我们简要介绍这种模式。

代理事件模型具有以下的主要参与者，即：

- 源 - 源是一个对象，在该对象上的事件发生。它的处理器提供发生事件的信息来源是可靠的。 JAVA提供源对象的类。
- 监听器 - 它也被称为事件处理程序。监听器是负责生成响应事件。从Java实现的角度来看，监听器也是一个对象。等待，直到它接收到一个事件监听器。一旦收到事件，监听器进程的事件然后返回。

这种方法的好处是，用户界面逻辑完全分开，生成该事件的逻辑。用户界面元素是能够代理的事件处理单独的一段代码。在这个模型中，需要与源对象注册监听使侦听器能够接收事件通知。这是一个有效的方式处理事件，因为这些事件通知只发送给那些监听器想要它们接收。

## 参与事件处理的步骤

- 用户单击该按钮时产生该事件。

- 现在有关事件类的对象是自动创建的信息源和事件在同一对象得到填充。
- 事件对象被转发注册监听器类的方法。
- 该方法现在得到执行和返回。

# 要记住的有关监听器要点

- 为了设计一个监听类，我们必须制定一些监听器接口。这些监听器接口预测一些公共的抽象监听器类必须实现的回调方法。
- 如果不实现任何预定义的接口，你的类不能作为源对象的监听器类。

# 回调方法

这些方法所提供的API提供者，被定义为应用程序员和应用程序开发者调用。这里的回调方法代表一个事件的方法。在响应一个事件的Java JRE将触发回调方法。所有这些回调方法的监听器接口。

如果一个组件需要一些监听器将监听的事件源必须注册自己的监听器。

# 事件处理示例

选择使用任何编辑器创建以下java程序 D:/ > AWT > com > yiibai > gui >

AwtControlDemo

```
package com.yiibai.gui;

import java.awt.*;
import java.awt.event.*;

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showEventDemo();
   }
```

```
   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showEventDemo(){
      headerLabel.setText("Control in action: Button");

      Button okButton = new Button("OK");
      Button submitButton = new Button("Submit");
      Button cancelButton = new Button("Cancel");

      okButton.setActionCommand("OK");
      submitButton.setActionCommand("Submit");
      cancelButton.setActionCommand("Cancel");

      okButton.addActionListener(new ButtonClickListener());
      submitButton.addActionListener(new ButtonClickListener());
      cancelButton.addActionListener(new ButtonClickListener());

      controlPanel.add(okButton);
      controlPanel.add(submitButton);
      controlPanel.add(cancelButton);

      mainFrame.setVisible(true);
   }

   private class ButtonClickListener implements ActionListener{
      public void actionPerformed(ActionEvent e) {
         String command = e.getActionCommand();
         if( command.equals( "OK" ))  {
            statusLabel.setText("Ok Button clicked.");
         }
         else if( command.equals( "Submit" ) )  {
```

```
                statusLabel.setText("Submit Button clicked.");
            }
            else  {
                statusLabel.setText("Cancel Button clicked.");
            }
        }
    }
}
```

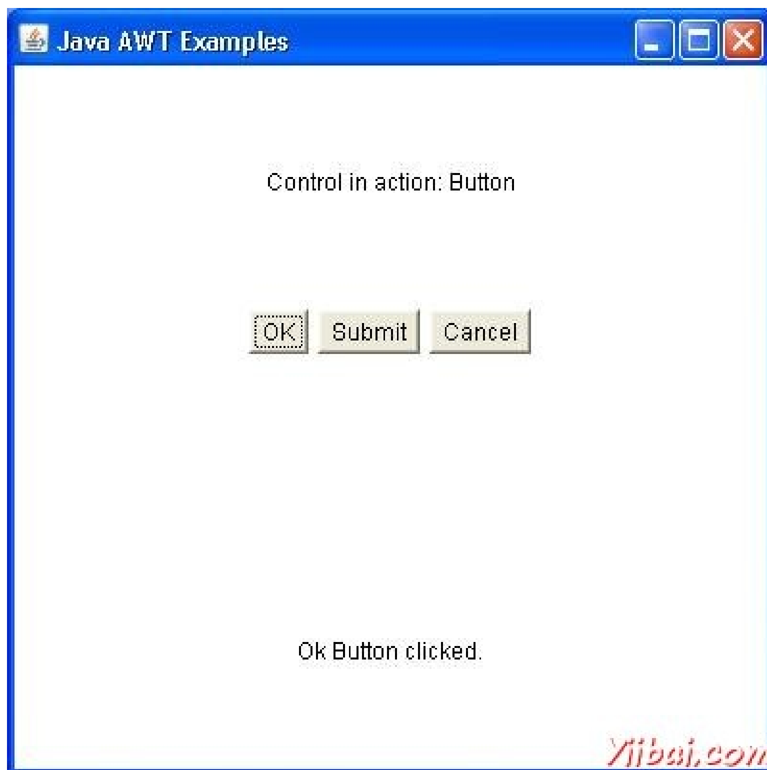
编译程序，使用命令提示符。到 **D:/ > AWT** 然后键入以下命令。

```
D:AWT>javac comyiibaiguiAwtControlDemo.java
```

如果没有错误出现，这意味着编译成功。使用下面的命令来运行程序。

```
D:AWT>java com.yiibai.gui.AwtControlDemo
```

验证下面的输出



# AWT Event类 - AWT

事件类代表了本次活动。 Java提供了各种事件类，但我们将讨论那些被更频繁地使用。

## EventObject 类

它是根类，应得到所有事件状态对象。构造一个对象的引用，所有事件的源，这是逻辑上被认为是事件最初发生时的对象。这个类定义在java.util包。

## 类的声明

以下是声明为java.util.EventObject类：

```
public class EventObject
   extends Object
      implements Serializable
```

## 字段域

以下是类java.util.EventObject 的字段：

- protected Object source -- 在其最初的事件发生的对象。

## 类的构造函数

| S.N. | 构造函数&说明 |
|---|---|
| 1 | **EventObject(Object source)** Constructs a prototypical Event. |

## 类方法

| S.N. | 方法和说明 |
|---|---|
| 1 | **Object getSource()** The object on which the Event initially occurred. |
| 2 | **String toString()** Returns a String representation of this EventObject. |

# 继承的方法

这个类从以下类继承的方法：

- java.lang.Object

# AWT事件类 ：

以下是常用的事件类的列表。

| Sr.No. | 控制与说明 |
| --- | --- |
| 1 | AWTEvent 它是所有AWT事件的根事件类。这个类及其子类取代原来的java.awt.Event类。 |
| 2 | ActionEvent 按钮被点击时，都会生成ActionEvent双击或列表中的项目。 |
| 3 | InputEvent InputEvent类是所有组件级别输入事件的根事件类。 |
| 4 | KeyEvent 输入字符键事件产生。 |
| 5 | MouseEvent 此事件表明在一个组件中发生鼠标动作。 |
| 6 | TextEvent 这个类的对象表示文本事件。 |
| 7 | WindowEvent 这个类的对象表示一个窗口的状态的变化。 |
| 8 | AdjustmentEvent 这个类的对象代表可调对象发出的调整事件。 |
| 9 | ComponentEvent 这个类的对象表示一个窗口的状态的变化。 |
| 10 | ContainerEvent 这个类的对象表示一个窗口的状态的变化。 |
| 11 | MouseMotionEvent 这个类的对象表示一个窗口的状态的变化。 |
| 12 | PaintEvent 这个类的对象表示一个窗口的状态的变化。 |

# AWT 事件监听器（Event Listeners） - AWT

事件侦听器代表负责处理事件的接口。Java提供了各种事件监听器类，但我们将讨论那些被更频繁地使用。每一个事件侦听器方法具有方法的EventObject类的子类的对象，这是作为一个单独的参数。例如，鼠标事件侦听器方法将接受MouseEvent的实例，其中派生的事件的EventObject。

## EventListner接口

它是一个标记接口，每一个监听器接口扩展。这个类定义在java.util包。

### 类的声明

以下是声明java.util.EventListener 接口：

```
public interface EventListener
```

## AWT Event Listener 接口:

以下是常用的事件侦听器列表。

| Sr. No. | 控制＆说明 |
|---|---|
| 1 | ActionListener 该接口用于接收动作事件。 |
| 2 | ComponentListener 该接口用于接收组件事件。 |
| 3 | ItemListener 该接口用于接收项目事件。 |
| 4 | KeyListener 该接口用于接收键事件。 |
| 5 | MouseListener 该接口用于接收鼠标事件。 |
| 6 | TextListener 该接口用于接收文本事件。 |
| 7 | WindowListener 该接口用于接收窗口事件。 |
| 8 | AdjustmentListener 该接口用于接收调整事件。 |
| 9 | ContainerListener 该接口用于接收容器事件。 |
| 10 | MouseMotionListener 此接口用于接收鼠标移动事件。 |
| 11 | FocusListener 该接口用于接收焦点事件。 |

# AWT 适配器(Adapters) - AWT

适配器是抽象类，用于接收各种事件。这些类中的方法是空的。这些类存在的目的是方便创建侦听器对象。

## AWT 适配器:

以下是常用的适配器监听AWT GUI事件列表。

| Sr. No. | 适配器＆说明 |
|---|---|
| 1 | FocusAdapter 接收焦点事件的抽象适配器类。 |
| 2 | KeyAdapter 接收按键事件的抽象适配器类。 |
| 3 | MouseAdapter 接收鼠标事件的抽象适配器类。 |
| 4 | MouseMotionAdapter 接收鼠标移动事件的抽象适配器类。 |
| 5 | WindowAdapter 接收窗口事件的抽象适配器类。 |

# AWT 布局（Layouts） - AWT

## 介绍

布局的意味着，在容器内的配置的组件。在其他的方式，我们可以说，在一个特定的容器内的位置放置组件。布局管理器所控制布点的任务是自动完成的。

## 布局管理器

布局管理器自动定位容器内的所有组件。如果我们不使用布局管理器，然后定位组件的默认布局管理器。这是可能的手工布局的控制，但由于以下两个原因，它变得非常困难。

- 这是非常繁琐的容器内处理大量的控制。
- 通常当我们需要来排列它们没有给出一个组件的宽度和高度信息。

Java为我们提供了各种布局管理器来定位控制。属性，如大小，形状和排列变化从一个布局管理器，其他的布局管理器。的小应用程序或应用程序窗口的大小改变时，即布局管理器applet浏览器或应用程序窗口的尺寸适应于响应的大小，形状和排列的组件也随之变化。

布局管理器关联的与每个容器对象。每一个布局管理器是实现布局管理接口的类的一个对象。

以下是接口定义布局管理器的功能。

| Sr. No. | 接口与说明 |
|---|---|
| 1 | LayoutManager LayoutManager 接口声明了类，其对象将充当一个布局管理器需要实现这些方法。 |
| 2 | LayoutManager2 LayoutManager2中的子接口布局管理。这个接口是为那些知道如何布局容器的基础上布局约束对象的类。 |

## AWT布局管理器类 :

以下是常用的控件列表而设计的图形用户界面使用AWT。

| Sr. No. | 布局管理说明 |
|---|---|
| 1 | BorderLayout BorderLayout 排列组件，以适应在五个区域：东部，西部，北部，南部和中心。 |
| 2 | CardLayout CardLayout对象将卡片作为一个容器中的每个组件。在一个时间只有一个卡片是可见的。 |
| 3 | FlowLayout FlowLayout将是默认的布局。它的布局有向流中的组件。 |
| 4 | GridLayout GridLayout 管理组件的矩形网格的形式。 |
| 5 | GridBagLayout 这是最灵活的布局管理器类。 GridBagLayout中的对象对齐的组件垂直方向，水平方向或沿它们的基线相同的大小，而不需要的组件。 |

# AWT 容器（Containers） - AWT

容器是AWT GUI组件的组成部分。其中一个组件可以位于容器提供空间。 AWT 容器本身是一个组件，它增加了添加组件本身的能力。以下要考虑要点。

- Container 的子类被称为容器。例如面板，框架和窗口。
- 容器只能添加组件本身。
- 默认的布局使用每个容器的setLayout方法可以覆盖使用。

| Sr. No. | 容器和说明 |
|---|---|
| 1 | Container 这是一个通用的容器对象中，可以包含其他AWT组件。 |

## AWT的UI元素：

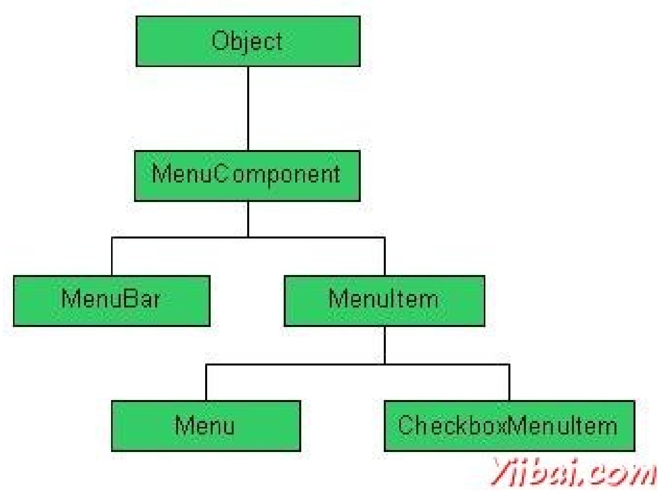
以下是常用的容器而设计的图形用户界面使用AWT。

| Sr. No. | 容器和说明 |
|---|---|
| 1 | Panel 面板是最简单的容器。它提供了空间，在其中可以放置任何其他组件，包括其他面板。 |
| 2 | Frame Frame是一个顶层的窗口标题和边框 |
| 3 | Window Window对象是一个没有边界和菜单栏的顶层窗口 |

# AWT菜单控制 - AWT

正如我们所知道相关的每一个顶层窗口有一个菜单栏。此菜单栏包括提供给最终用户的各种菜单的选择。此外，每个选择包含这就是所谓的下拉菜单中的选项列表。菜单和菜单项的控件是的MenuComponent类的子类。

## 菜单层次结构




## 菜单控件

| Sr. No. | 控件与说明 |
|---|---|
| 1 | MenuComponent 它是顶级类的所有菜单相关的控制。 |
| 2 | MenuBar MenuBar对象相关联的顶层窗口。 |
| 3 | MenuItem 菜单中的项目必须属于MenuItem或任何其子类。 |
| 4 | Menu Menu对象是从菜单栏中显示一个下拉菜单组件。 |
| 5 | CheckboxMenuItem CheckboxMenuItem是菜单项的子类。 |
| 6 | PopupMenu PopupMenu 弹出式菜单，可以在一个组件内的指定位置动态弹出。 |

# AWT图形控件 - AWT

图形控件允许应用程序组件或图像上绘制。

## 图形控件

| Sr. No. | 控制与说明 |
| --- | --- |
| 1 | Graphics 这是所有图形上下文顶层抽象类。 |
| 2 | Graphics2D 这是一个Graphics类的子类，并提供了更复杂的几何坐标变换，色彩管理，和文本布局的控制权。 |
| 3 | Arc2D Arc2D是抽象父类的所有对象存储框架矩形定义一个二维弧，开始角度，棱角分明的程度（弧长）和闭合类型（OPEN，CHORD或PIE）。 |
| 4 | CubicCurve2D CubicCurve2D类是抽象的超类FPR存储2D三次曲线段的所有对象，它定义的三次参数曲线段（X，Y）坐标空间。 |
| 5 | Ellipse2D Ellipse2D的是抽象超类的所有对象存储2D椭圆形，它描述了一种椭圆形，是指由一划分矩形。 |
| 6 | Rectangle2D Rectangle2D类是一个抽象父类的所有对象存储2D矩形所定义的位置（X，Y）和尺寸（宽x高），它描述了一个矩形。 |
| 7 | QuadCurve2D QuadCurve2D类是一个抽象父类的所有对象存储2D二次曲线段，它描述了一个二次参数曲线段（X，Y）坐标空间。 |
| 8 | Line2D Line2D表示（的x，y）坐标空间中的线段。 |
| 9 | Font Font类表示字体，它是在一个可见的方式呈现文本。 |
| 10 | Color Color类用于封装默认sRGB颜色空间或颜色的任意颜色空间中的颜色标识的颜色。 |
| 11 | BasicStroke BasicStroke的类定义了一套基本的轮廓基本图形，这是一个Graphics2D对象有其行程的属性设置与此BasicStroke呈现的渲染属性。 |

# EasyMock教程

## Mocking是什么?

Mocking是一种在隔离测试一个类的功能。例如，无需数据库连接或属性文件中读取或文件服务器上读取需要测试的功能。 mock对象做实服务的嘲讽。一个mock对象返回对应于传递给它一些虚拟输入无效数据。

## EasyMock

EasyMock便于无缝地创建模拟对象。它使用Java反射，以创造为给定接口的模拟对象。模拟对象是什么，只不过是代理的实际实现。考虑如：股票服务的情况下，它返回一个股票价格的详细信息。在开发过程中，实际的库存服务不能被用于获得实时数据。因此，我们需要一个虚拟的股票实施服务。简易模拟可以很容易理解顾名思义这样。

## EasyMock的好处

- 不用手写 - 没有必要通过自己编写的模拟对象。
- 重构安全 - 重构接口方法的名称或重新排序的参数不会破坏测试代码在运行时创建。
- 返回值支持 - 支持返回值。
- 异常支持 - 支持例外/异常。
- 命令检查支持 - 支持检查命令方法调用。
- 注释支持 - 支持使用注解创建。

考虑下面的代码片段。

```
package com.yiibai.mock;

import java.util.ArrayList;
import java.util.List;

import org.easymock.EasyMock;

public class PortfolioTester {
   public static void main(String[] args){

      //Create a portfolio object which is to be tested
      Portfolio portfolio = new Portfolio();

      //Creates a list of stocks to be added to the portfolio
      List<Stock> stocks = new ArrayList<Stock>();
      Stock googleStock = new Stock("1","Google", 10);
      Stock microsoftStock = new Stock("2","Microsoft",100);

      stocks.add(googleStock);
      stocks.add(microsoftStock);

      //Create the mock object of stock service
      StockService stockServiceMock = EasyMock.createMock(StockServ

      //mock the behavior of stock service to return the value of v
      EasyMock.expect(stockServiceMock.getPrice(googleStock)).andRe
      EasyMock.expect(stockServiceMock.getPrice(microsoftStock)).an

      EasyMock.replay(stockServiceMock);
      //add stocks to the portfolio
      portfolio.setStocks(stocks);
      //set the stockService to the portfolio
      portfolio.setStockService(stockServiceMock);

      double marketValue = portfolio.getMarketValue();

      //verify the market value to be 10*50.00 + 100* 1000.00 = 500
      System.out.println("Market value of the portfolio: "+ marketV
   }
}
```

让我们来了解上述程序的重要概念。完整的代码在第一个应用。

- **Portfolio** - 进行股票名单，并获得用股票价格和股票数量计算的市场价值的对象。
- **Stock** - 携带一只股票的详细信息，如它的id，名称，数量等的对象
- **StockService** - 股票的服务接口，其功能是返回一个股票的当前价格。
- **EasyMock.createMock(...)** - EasyMock股票创建了服务的模拟

- **EasyMock.expect(...).andReturn(...)** - 模拟实现StockService接口用getPrice方法。对于googleStock，回到50.00的价格。
- **EasyMock.replay(...)** - EasyMock准备模拟对象，以便它可以被用于测试目的。
- **portfolio.setStocks(...)** - 现在的投资组合包含了两只股票列表。
- **portfolio.setStockService(...)** - 分配的StockService模拟对象来组合。
- **portfolio.getMarketValue()** - 基于使用的模拟股票服务公司的股票投资组合回报的市场价值。

# EasyMock环境安装 - EasyMock教程

EasyMock是Java框架，所以第一个要求是要在机器安装JDK。

## 系统要求

| JDK | 1.5 或以上. |
|---|---|
| 内存 | 没有最低要求 |
| 硬盘空间 | 没有最低要求 |
| 操作系统 | 没有最低要求 |

## 第1步 - 验证Java安装在机器

现在，打开控制台并执行以下java命令。

| OS | 任务 | 命令 |
|---|---|---|
| Windows | 打开命令控制台 | c:\> java -version |
| Linux | 打开命令终端 | $ java -version |
| Mac | 打开终端 | machine:~ joseph$ java -version |

让我们来为所有的操作系统验证输出：

Windows

```
java version "1.6.0_21"
Java(TM) SE Runtime Environment (build 1.6.0_21-b07)
Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
```

Linux

```
java version "1.6.0_21"
Java(TM) SE Runtime Environment (build 1.6.0_21-b07)
Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
```

Mac

```
java version "1.6.0_21"
Java(TM) SE Runtime Environment (build 1.6.0_21-b07)
Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, shar
```

如果还没有安装Java，安装Java软件开发工具包(SDK)可从 http://www.oracle.com/technetwork/java/javase/downloads/index.htmll 下载最新版本。我们假设Java1.6.0_21 作为本教程的安装版本。

## 第2步：设置JAVA环境

设置 JAVA_HOME 环境变量指向到安装在机器上的Java的基本目录的位置。例如

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 JAVA_HOME 到 C:\Program Files\Java\jdk1.6.0_21 |
| Linux | export JAVA_HOME=/usr/local/java-current |
| Mac | export JAVA_HOME=/Library/Java/Home |

添加 Java编译器位置到系统路径。

| OS | 输出 |
|---|---|
| Windows | Append the string ;C:\Program Files\Java\jdk1.6.0_21\bin to the end of the system variable, Path. |
| Linux | export PATH=$PATH:$JAVA_HOME/bin/ |
| Mac | not required |

用 java -version 命令验证如上所述Java安装。

## 第3步：下载EasyMock

从 http://sourceforge.net/projects/easymock/files/EasyMock/3.2/easymock-3.2.zip/download 下载EasyMock的zip文件的最新版本。在写这篇教程的时候，下载的是 easymock-3.2.zip 并将其复制到C:\>EasyMock 文件夹。

| OS | 归档文件名称 |
|---|---|
| Windows | easymock-3.2.zip |
| Linux | easymock-3.2.zip |
| Mac | easymock-3.2.zip |

# 第4步：下载EasyMock的依赖包

从https://github.com/cglib/cglib/releases下载cglib的jar文件的最新版本。在写这篇教程的时候，我下载的是 cglib-3.1.jar 并将其复制到 C：\> EasyMock 文件夹。

从http://objenesis.org/download.htmll下载objenesis.zip文件的最新版本。在写这篇教程的时候，我下载objenesis-2.1-bin.zip 并将其复制到C：\> EasyMock文件夹。提取objenesis-2.1.jar到C：\> EasyMock文件夹

# 第5步：设置EasyMock的环境

设置EasyMock_HOME环境变量指向的地方EasyMock的和依赖的jar都存储在您计算机上的基本目录的位置。假设，我们已经提取了easymock-3.2.jar, cglib-3.1.jar 和objenesis-2.1.jar在各种操作系统上EasyMock的文件夹，如下所示。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量EasyMock_HOME to C:\EasyMock |
| Linux | export EasyMock_HOME=/usr/local/EasyMock |
| Mac | export EasyMock_HOME=/Library/EasyMock |

# 第5步：设置CLASSPATH变量

设置CLASSPATH环境变量指向了EasyMock和依赖的jar位置。假设，我们已经存储了easymock-3.2.jar, cglib-3.1.jar 和objenesis-2.1.jar i在EasyMock的文件夹，在各种操作系统上，如下所示。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量CLASSPATH to %CLASSPATH%;%EasyMock_HOME%\easymock-3.2.jar;%EasyMock_HOME%\cglib-3.1.jar;%EasyMock_HOME%\objenesis-2.1.jar;.; |
| Linux | export CLASSPATH=$CLASSPATH:$EasyMock_HOME/easymock-3.2.jar:$EasyMock_HOME/cglib-3.1.jar:$EasyMock_HOME/objenesis-2.1.jar:. |
| Mac | export CLASSPATH=$CLASSPATH:$EasyMock_HOME/easymock-3.2.jar:$EasyMock_HOME/cglib-3.1.jar:$EasyMock_HOME/objenesis-2.1.jar:. |

## 第6步：下载Junit的归档

从https://github.com/junit-team/junit/wiki/Download-and-Install下载JUnit的jar文件的最新版本。在写这篇教程的时候，我下载的是Junit-4.11.jar,hamcrest-core-1.2.1.jar，并复制它们到C：\> JUnit文件夹中。

| OS | 归档名称 |
|---|---|
| Windows | junit4.11.jar, hamcrest-core-1.2.1.jar |
| Linux | junit4.11.jar, hamcrest-core-1.2.1.jar |
| Mac | junit4.11.jar, hamcrest-core-1.2.1.jar |

## 第7步：设置JUnit的环境

设置JUNIT_HOME环境变量指向JUNIT的jar都存储在您计算机上的基本目录的位置。假设，我们已经存储junit4.11.jar, hamcrest-core-1.2.1.jar 在JUnit文件夹在各种操作系统上，如下所示。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 JUNIT_HOME to C:\JUNIT |
| Linux | export JUNIT_HOME=/usr/local/JUNIT |
| Mac | export JUNIT_HOME=/Library/JUNIT |

## 第8步：设置CLASSPATH变量

设置CLASSPATH环境变量指向了JUnit的jar位置。假设，我们已经存储junit4.10.jar在JUnit夹在各种操作系统上，如下所示。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 CLASSPATH 为 %CLASSPATH%;%JUNIT_HOME%\junit4.11.jar;%JUNIT_HOME%\ 1.2.1.jar;.; |
| Linux | export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.11.jar:$JUNIT_H core-1.2.1.jar:. |
| Mac | export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.11.jar:$JUNIT_H core-1.2.1.jar:. |

# EasyMock异常处理 - EasyMock教程

EasyMock提供了一个功能，用以模拟抛出异常，所以异常处理可以进行测试。

```
//add the behavior to throw exception
EasyMock.expect(calcService.add(10.0,20.0)).andThrow(new RuntimeExc
```

在这里，我们添加了一个异常子句模仿对象。 MathApplication利用使用calcService其Add方法和模型将抛出RuntimeException异常时调用calcService.add()方法。

## 示例

创建一个接口CalculatorService，其目的是提供各种计算相关的功能。

*CalculatorService.java*

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

创建一个Java类用来表示MathApplication。

*MathApplication.java*

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }
   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }
   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }
   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }
   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

让我们来测试MathApplication类，通过它注入CalculatorService作一个模拟。Mock将由EasyMock创建。

*MathApplicationTester.java*

```
import org.easymock.EasyMock;
import org.easymock.EasyMockRunner;
import org.easymock.Mock;
import org.easymock.TestSubject;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;

//@RunWith attaches a runner with the test class to initialize the
@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

   //@TestSubject annotation is used to identify class which is goi
   //mock object
   @TestSubject
   MathApplication mathApplication = new MathApplication();

   //@Mock annotation is used to create the mock object to be injec
   @Mock
   CalculatorService calcService;

   @Test(expected = RuntimeException.class)
   public void testAdd(){
      //add the behavior to throw exception
      EasyMock.expect(calcService.add(10.0,20.0)).andThrow(new Runt
      //activate the mock
      EasyMock.replay(calcService);
      //test the add functionality
      Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);
      //verify call to calcService is made or not
      EasyMock.verify(calcService);
   }
}
```

创建一个Java类在文件夹 C:\ > EasyMock_WORKSPACE 执行测试用例

*TestRunner.java*

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(MathApplicationTester.cl
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

## 验证结果

使用javac编译如下类

```
C:\EasyMock_WORKSPACE>javac MathApplicationTester.java
```

现在运行测试运行看结果

```
C:\EasyMock_WORKSPACE>java TestRunner
```

验证输出

```
true
```

# EasyMock JUnit集成 - EasyMock教程

在本章中，我们将集成JUnit和EasyMock在一起。对于JUnit，请参阅JUnit教程。我们使用计算器服务的例子。目的是创建一个数学应用，它使用CalculatorService做加，减，除运算操作。我们将使用EasyMock来模拟虚拟实现CalculatorService。此外，使用注解广泛展示注解支持JUnit和EasyMock。

以下是所采取的步骤。

创建一个接口CalculatorService，其目的是提供各种计算相关的功能。

CalculatorService.java

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

创建一个Java类来表示MathApplication.

MathApplication.java

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService)·
      this.calcService = calcService;
   }
   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }
   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }
   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }
   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

让我们来测试MathApplication类，通过它注入CalculatorService作一个模拟。Mock将由EasyMock创建。

MathApplicationTester.java

```
import org.easymock.EasyMock;
import org.easymock.EasyMockRunner;
import org.easymock.Mock;
import org.easymock.TestSubject;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

//@RunWith attaches a runner with the test class to initialize the
@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

   //@TestSubject annotation is used to identify class which is goi
   //mock object
   @TestSubject
   MathApplication mathApplication = new MathApplication();

   //@Mock annotation is used to create the mock object to be injec
   @Mock
   CalculatorService calcService;

   @Test
   public void testAdd(){
      //add the behavior of calc service to add two numbers
      EasyMock.expect(calcService.add(10.0,20.0)).andReturn(30.00);

      //activate the mock
      EasyMock.replay(calcService);

      //test the add functionality
      Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);
   }
}
```

创建一个Java类在文件夹 C:\ > EasyMock_WORKSPACE 执行测试用例

TestRunner.java

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(MathApplicationTester.cl
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

# 验证结果

编译使用javac编译如下的类

```
C:\EasyMock_WORKSPACE>javac CalculatorService.java MathApplication
```

现在运行测试运行看结果

```
C:\EasyMock_WORKSPACE>java TestRunner
```

验证输出

```
true
```

# EasyMock添加行为 - EasyMock教程

EasyMock使用expect()方法或expectLassCall()方法添加一个功能，一个模拟对象。请看下面的代码片段。

```
//add the behavior of calc service to add two numbers
EasyMock.expect(calcService.add(10.0,20.0)).andReturn(30.00);
```

这里，我们已经指示EasyMock，行为添加10和20到calcService的添加方法并作为其结果，到返回值30.00

在这个时间点上，模拟简单记录的行为，但它本身不作为一个模拟对象。调用回放后，按预期工作。

```
//add the behavior of calc service to add two numbers
EasyMock.expect(calcService.add(10.0,20.0)).andReturn(30.00);

//activate the mock
//EasyMock.replay(calcService);
```

不需要EasyMock.Replay()的示例

需要EasyMock.Replay()的示例

# EasyMock验证行为 - EasyMock教程

EasyMock提供了一个检查被使用或不使用verify()方法，请看下面的代码片段。

```
//activate the mock
EasyMock.replay(calcService);

//test the add functionality
Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);

//verify call to calcService is made or not
EasyMock.verify(calcService);
```

不使用EasyMock.Verify()示例

使用 EasyMock.Verify()示例

# EasyMock期望调用 - EasyMock教程

EasyMock提供，可以在特定的方法来的调用的数目的特别检查。假设MathApplication使用其的任意方法，其中CalculatorService.serviceUsed()方法表示CalculatorService的用于获得所需要的操作结果之前调用CalculatorService.serviceUsed()方法，只有一次。 MathApplication应该不能够调用CalculatorService.serviceUsed()一次以上。

```
//add the behavior of calc service to add two numbers and serviceUs
EasyMock.expect(calcService.add(10.0,20.0)).andReturn(30.00);
calcService.serviceUsed();
//limit the method call to 1, no less and no more calls are allowed
EasyMock.expectLastCall().times(1);
```

创建CalculatorService的界面如下。

CalculatorService.java

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
   public void serviceUsed();
}
```

calcService.serviceUsed()被调用一次例子

calcService.serviceUsed()调用两次例子

无需调用calcService.serviceUsed()示例

# EasyMock改变调用 - EasyMock教程

EasyMock提供很多的方法来改变预期的调用计数。

- times(int min, int max) - 最小值和最大值之间的预期调用。
- atLeastOnce() - 预期至少有一个调用。
- anyTimes() - 预期调用的数量不受限制。

times(min,max)的例子

atLeastOnce例子

anyTimes例子

# EasyMock异常处理 - EasyMock教程

EasyMock提供了一个功能，用以模拟抛出异常，所以异常处理可以进行测试。

```
//add the behavior to throw exception
EasyMock.expect(calcService.add(10.0,20.0)).andThrow(new RuntimeExc
```

在这里，我们添加了一个异常子句模仿对象。 MathApplication利用使用calcService其Add方法和模型将抛出RuntimeException异常时调用calcService.add()方法。

## 示例

创建一个接口CalculatorService，其目的是提供各种计算相关的功能。

*CalculatorService.java*

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

创建一个Java类用来表示MathApplication。

*MathApplication.java*

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }
   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }
   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }
   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }
   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

让我们来测试MathApplication类，通过它注入CalculatorService作一个模拟。Mock将由EasyMock创建。

*MathApplicationTester.java*

```
import org.easymock.EasyMock;
import org.easymock.EasyMockRunner;
import org.easymock.Mock;
import org.easymock.TestSubject;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;

//@RunWith attaches a runner with the test class to initialize the
@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

   //@TestSubject annotation is used to identify class which is goi
   //mock object
   @TestSubject
   MathApplication mathApplication = new MathApplication();

   //@Mock annotation is used to create the mock object to be injec
   @Mock
   CalculatorService calcService;

   @Test(expected = RuntimeException.class)
   public void testAdd(){
      //add the behavior to throw exception
      EasyMock.expect(calcService.add(10.0,20.0)).andThrow(new Runt
      //activate the mock
      EasyMock.replay(calcService);
      //test the add functionality
      Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);
      //verify call to calcService is made or not
      EasyMock.verify(calcService);
   }
}
```

创建一个Java类在文件夹 C:\ > EasyMock_WORKSPACE 执行测试用例

*TestRunner.java*

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(MathApplicationTester.cl
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

# 验证结果

使用javac编译如下类

```
C:\EasyMock_WORKSPACE>javac MathApplicationTester.java
```

现在运行测试运行看结果

```
C:\EasyMock_WORKSPACE>java TestRunner
```

验证输出

```
true
```

# EasyMock createMock - EasyMock教程

到目前为止，我们已经使用注解来创建Mocks。 EasyMock提供了各种方法来创建模拟对象。 EasyMock.createMock()创建的模拟，但没有理会方法的顺序调用模拟会在作出其行动的适当时机。

## 语法

```
calcService = EasyMock.createMock(CalculatorService.class);
```

## 示例

创建一个接口CalculatorService，其目的是提供各种计算相关的功能。

*CalculatorService.java*

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

创建一个Java类用来表示MathApplication。

*MathApplication.java*

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }
   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }
   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }
   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }
   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

让我们来测试MathApplication类，通过它注入CalculatorService作一个模拟。Mock将由EasyMock创建。

在这里，我们已经添加了两个模拟方法调用，add()和subtract()来模拟对象，通过expect()。但在测试过程中，我们调用Add()方法前调用subtract()。当我们创建模拟对象使用EasyMock.createMock()，以便执行方法。

*MathApplicationTester.java*

```
import org.easymock.EasyMock;
import org.easymock.EasyMockRunner;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

   private MathApplication mathApplication;
   private CalculatorService calcService;

   @Before
   public void setUp(){
      mathApplication = new MathApplication();
      calcService = EasyMock.createMock(CalculatorService.class);
      mathApplication.setCalculatorService(calcService);
   }

   @Test
   public void testAddAndSubstract(){
      //add the behavior to add numbers
      EasyMock.expect(calcService.add(20.0,10.0)).andReturn(30.0);
      //subtract the behavior to subtract numbers
      EasyMock.expect(calcService.subtract(20.0,10.0)).andReturn(1(
      //activate the mock
      EasyMock.replay(calcService);
      //test the substract functionality
      Assert.assertEquals(mathApplication.subtract(20.0, 10.0),10.(
      //test the add functionality
      Assert.assertEquals(mathApplication.add(20.0, 10.0),30.0,0);
      //verify call to calcService is made or not
      EasyMock.verify(calcService);
   }
}
```

创建一个Java类在文件夹 C:\ > EasyMock_WORKSPACE 执行测试用例

*TestRunner.java*

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(MathApplicationTester.cl
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

## 验证结果

使用javac编译如下类

```
C:\EasyMock_WORKSPACE>javac MathApplicationTester.java
```

现在运行测试运行看结果

```
C:\EasyMock_WORKSPACE>java TestRunner
```

验证输出

```
true
```

# EasyMock createStrictMock - EasyMock教程

EasyMock.createStrictMock()创建的模拟，也需要关心的方法的顺序调用模拟会在作出其行动的适当时机。

## 语法

```
calcService = EasyMock.createStrictMock(CalculatorService.class);
```

## 示例

创建一个接口CalculatorService，其目的是提供各种计算相关的功能。

*CalculatorService.java*

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

创建一个Java类用来表示MathApplication。

*MathApplication.java*

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }
   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }
   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }
   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }
   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

让我们来测试MathApplication类，通过它注入CalculatorService作一个模拟。Mock将由EasyMock创建。

在这里，我们已经添加了两个模拟方法调用，add()和subtract()来模拟对象，通过expect()。但在测试过程中，我们调用Add()方法前调用subtract()。当我们创建一个使用EasyMock.createStrictMock()的模拟对象，以便执行方法。

*MathApplicationTester.java*

```
import org.easymock.EasyMock;
import org.easymock.EasyMockRunner;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

   private MathApplication mathApplication;
   private CalculatorService calcService;

   @Before
   public void setUp(){
      mathApplication = new MathApplication();
      calcService = EasyMock.createStrictMock(CalculatorService.cla
      mathApplication.setCalculatorService(calcService);
   }

   @Test
   public void testAddAndSubstract(){
      //add the behavior to add numbers
      EasyMock.expect(calcService.add(20.0,10.0)).andReturn(30.0);
      //subtract the behavior to subtract numbers
      EasyMock.expect(calcService.subtract(20.0,10.0)).andReturn(10
      //activate the mock
      EasyMock.replay(calcService);
      //test the substract functionality
      Assert.assertEquals(mathApplication.subtract(20.0, 10.0),10.0
      //test the add functionality
      Assert.assertEquals(mathApplication.add(20.0, 10.0),30.0,0);
      //verify call to calcService is made or not
      EasyMock.verify(calcService);
   }
}
```

创建一个Java类在文件夹 C:\ > EasyMock_WORKSPACE 执行测试用例

*TestRunner.java*

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(MathApplicationTester.cl
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

## 验证结果

使用javac编译如下类

```
C:\EasyMock_WORKSPACE>javac MathApplicationTester.java
```

现在运行测试运行看结果

```
C:\EasyMock_WORKSPACE>java TestRunner
```

验证输出

```
testAddAndSubstract(com.yiibai.mock.MathApplicationTester):
  Unexpected method call CalculatorService.subtract(20.0, 10.0):
    CalculatorService.add(20.0, 10.0): expected: 1, actual: 0
false
```

# EasyMock createNiceMock - EasyMock教程

EasyMock.createNiceMock()创建了模拟，并设置模拟的每个方法的默认实现。如果使用EasyMock.createMock()，然后模拟方法调用将抛出断言错误。

## 语法

```
calcService = EasyMock.createNiceMock(CalculatorService.class);
```

## 示例

创建一个接口CalculatorService，其目的是提供各种计算相关的功能。

*CalculatorService.java*

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

创建一个Java类用来表示MathApplication。

*MathApplication.java*

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }
   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }
   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }
   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }
   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

让我们来测试MathApplication类，通过它注入CalculatorService作一个模拟。Mock将由EasyMock创建。

在这里，我们已经增加了一个模拟的方法调用，add()通过expect()。但在测试过程中，我们称之为substract()等方法。当创建一个使用EasyMock.createNiceMock()模拟对象，用缺省值缺省实现是可用的。

*MathApplicationTester.java*

```
import org.easymock.EasyMock;
import org.easymock.EasyMockRunner;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

   private MathApplication mathApplication;

   private CalculatorService calcService;

   @Before
   public void setUp(){
      mathApplication = new MathApplication();
      calcService = EasyMock.createNiceMock(CalculatorService.class
      mathApplication.setCalculatorService(calcService);
   }

   @Test
      public void testCalcService(){
      //add the behavior to add numbers
      EasyMock.expect(calcService.add(20.0,10.0)).andReturn(30.0);
      //activate the mock
      EasyMock.replay(calcService);
      //test the add functionality
      Assert.assertEquals(mathApplication.add(20.0, 10.0),30.0,0);
      //test the substract functionality
      Assert.assertEquals(mathApplication.subtract(20.0, 10.0),0.0,
      //test the multiply functionality
      Assert.assertEquals(mathApplication.divide(20.0, 10.0),0.0,0)
      //test the divide functionality
      Assert.assertEquals(mathApplication.multiply(20.0, 10.0),0.0,

      //verify call to calcService is made or not
      EasyMock.verify(calcService);
   }
}
```

创建一个Java类在文件夹 C:\ > EasyMock_WORKSPACE 执行测试用例

*TestRunner.java*

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(MathApplicationTester.cl
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

## 验证结果

使用javac编译如下类

```
C:\EasyMock_WORKSPACE>javac MathApplicationTester.java
```

现在运行测试运行看结果

```
C:\EasyMock_WORKSPACE>java TestRunner
```

验证输出

```
true
```

# EasyMock EasyMockSupport - EasyMock教程

EasyMockSupport是Utiliy或辅助类的测试类。它提供了以下功能

- replayAll() - 都记录在一个批次中创建模拟。
- verifyAll() - 验证所有模拟操作于一个批次。
- resetAll() - 将所有模拟操作于一个批次。

## 示例

创建一个接口CalculatorService，其目的是提供各种计算相关的功能。

*CalculatorService.java*

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

创建一个Java类用来表示MathApplication。

*MathApplication.java*

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }
   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }
   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }
   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }
   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

让我们来测试MathApplication类，通过它注入CalculatorService作一个模拟。Mock将由EasyMock创建。

*MathApplicationTester.java*

```
import org.easymock.EasyMockRunner;
import org.easymock.EasyMockSupport;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(EasyMockRunner.class)
public class MathApplicationTester extends EasyMockSupport {

   private MathApplication mathApplication1;
   private MathApplication mathApplication2;

   private CalculatorService calcService1;
   private CalculatorService calcService2;

   @Before
   public void setUp(){
      mathApplication1 = new MathApplication();
      mathApplication2 = new MathApplication();
      calcService1 = createNiceMock(CalculatorService.class);
      calcService2 = createNiceMock(CalculatorService.class);
      mathApplication1.setCalculatorService(calcService1);
      mathApplication2.setCalculatorService(calcService2);
   }
```

```
   @Test
   public void testCalcService(){
      //activate all mocks
      replayAll();
      //test the add functionality
      Assert.assertEquals(mathApplication1.add(20.0, 10.0),0.0,0);
      //test the substract functionality
      Assert.assertEquals(mathApplication1.subtract(20.0, 10.0),0.0
      //test the multiply functionality
      Assert.assertEquals(mathApplication1.divide(20.0, 10.0),0.0,0
      //test the divide functionality
      Assert.assertEquals(mathApplication1.multiply(20.0, 10.0),0.0

      //test the add functionality
      Assert.assertEquals(mathApplication2.add(20.0, 10.0),0.0,0);
      //test the substract functionality
      Assert.assertEquals(mathApplication2.subtract(20.0, 10.0),0.0
      //test the multiply functionality
      Assert.assertEquals(mathApplication2.divide(20.0, 10.0),0.0,0
      //test the divide functionality
      Assert.assertEquals(mathApplication2.multiply(20.0, 10.0),0.0

      //verify all the mocks
      verifyAll();
   }
}
```

创建一个Java类在文件夹 C:\ > EasyMock_WORKSPACE 执行测试用例

*TestRunner.java*

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(MathApplicationTester.cl
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

# 验证结果

使用javac编译如下类

```
C:\EasyMock_WORKSPACE>javac MathApplicationTester.java
```

现在运行测试运行看结果

```
C:\EasyMock_WORKSPACE>java TestRunner
```

验证输出

```
true
```

# Eclipse 教程




Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。

Eclipse 是 Java 的集成开发环境（IDE），当然 Eclipse 也可以作为其他开发语言的集成开发环境，如C，C++，PHP，和 Ruby 等。

Eclipse 附带了一个标准的插件集，包括Java开发工具（Java Development Kit，JDK）。

## 许可证

Eclipse 平台及其插件基于Eclipse公共许可证（EPL：Eclipse Public License）发布。

Eclipse公共许可证（简称EPL）是一种开源软件发布许可证，由Eclipse基金会应用于名下的集成开发环境Eclipse上。

## 发行版

从2006年起，Eclipse基金会每年都会安排同步发布。至今，同步发布主要在6月进行，并且会在接下来的9月及2月释放出SR1及SR2版本。

| 版本代号 | 平台版本 | 主要版本发行日期 | SR1发行日期 | SR2发行日期 |
| --- | --- | --- | --- | --- |
| Callisto | 3.2 | 2006年6月26日 | N/A | N/A |
| Europa | 3.3 | 2007年6月27日 | 2007年9月28日 | 2008年2月29日 |
| Ganymede | 3.4 | 2008年6月25日 | 2008年9月24日 | 2009年2月25日 |
| Galileo | 3.5 | 2009年6月24日 | 2009年9月25日 | 2010年2月26日 |
| Helios | 3.6 | 2010年6月23日 | 2010年9月24日 | 2011年2月25日 |
| Indigo | 3.7 | 2011年6月22日 | 2011年9月23日 | 2012年2月24日 |
| Juno | 3.8及4.2 | 2012年6月27日 | 2012年9月28日 | 2013年3月1日 |
| Kepler | 4.3 | 2013年6月26日 | 2013年9月27日 | 2014年2月28日 |
| Luna | 4.4 | 2014年6月25日 | 2014年9月25日 | N/A |
| Mars | 4.5 | 2015年6月24日 | N/A | N/A |

# Eclipse 安装

下载 Eclipse，下载地址为：http://www.eclipse.org/downloads/。 下载页面列出了不同语言的Eclipse IDE，你可以根据自己需要下载。

Eclipse 的每个安装包都不同， Java 开发人员通常使用 Eclipse IDE for Java Developers 来开发 Java 应用。

列表右侧提供了Windows，Linux 和 Mac 操作系统及对应的32位与64位的安装包，你可以根据自己的系统情况选择合适的包下载。

Eclipse Luna (4.4.1) Release for Windows

**Eclipse IDE for Java Developers**, 154 MB
Downloaded 2,581,552 Times
The essential tools for any Java developer, including a Java IDE, a CVS client, Git client, XML Editor, Mylyn, Maven integration...
Windows 32 Bit
Windows 64 Bit

Package Solutions
Filter Packages

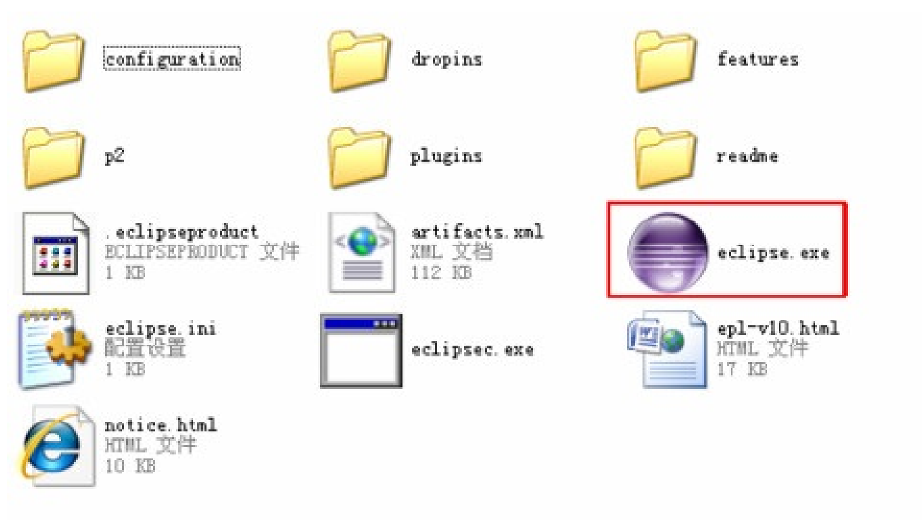
**Eclipse IDE for Java EE Developers**, 254 MB
Downloaded 1,570,255 Times
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...
Windows 32 Bit
Windows 64 Bit

**JRebel for Eclipse IDE**
See Java Code Changes Instantly. Save Time. Reduce Stress. Finish Projects Faster!
Promoted Download

# 安装 Eclipse

Eclipse 是基于 Java 的可扩展开发平台，所以安装 Eclipse 前你需要确保你的电脑已安装 JDK，JDK 安装可以查看我们的Java开发环境配置。

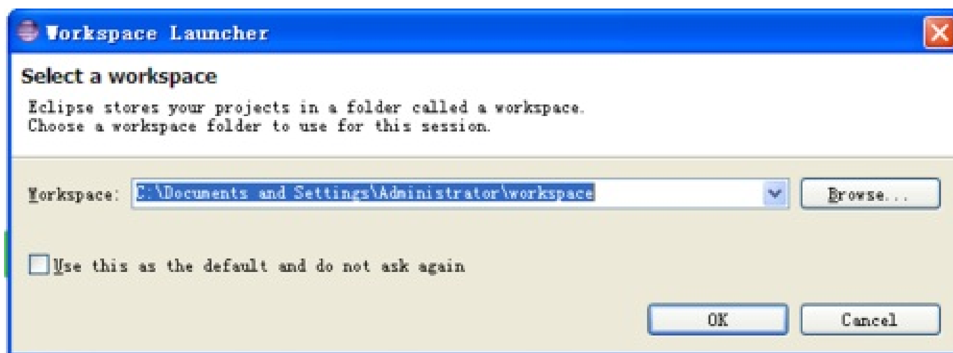
Eclipse 的安装非常简单，你只需要下载压缩包，解压完毕后即可使用，进入文件夹，红框如图所示就是eclipse的启动程序



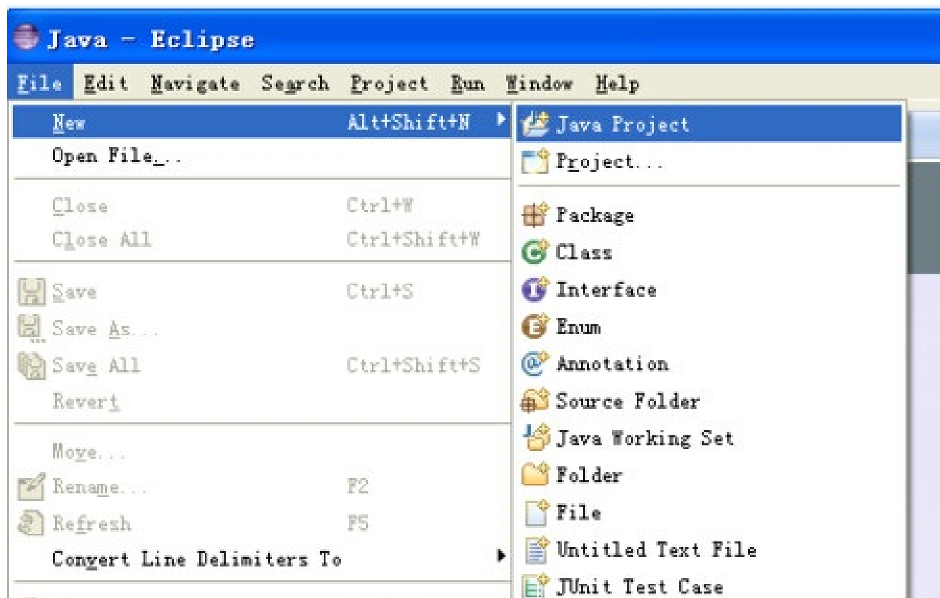

若你打开Eclipse的时候发现如下的对话框，则说明你的电脑未安装 JDK 环境。



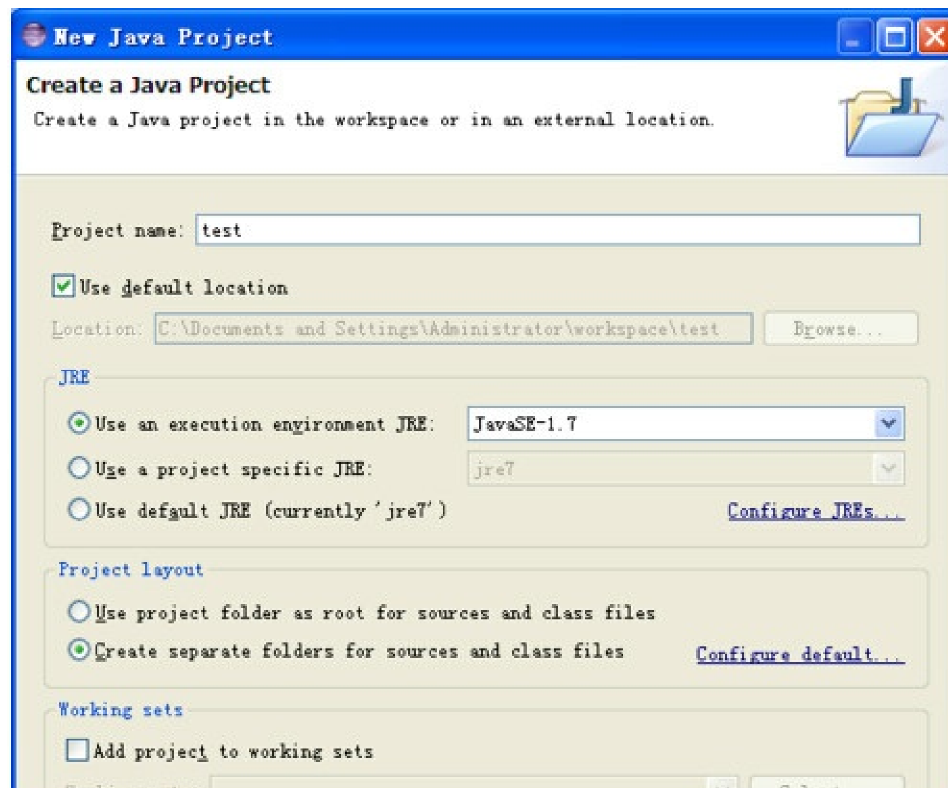

第一次打开需要设置工作环境，你可以指定工作目录，或者使用默认的C盘工作目录，点击 ok 按钮。



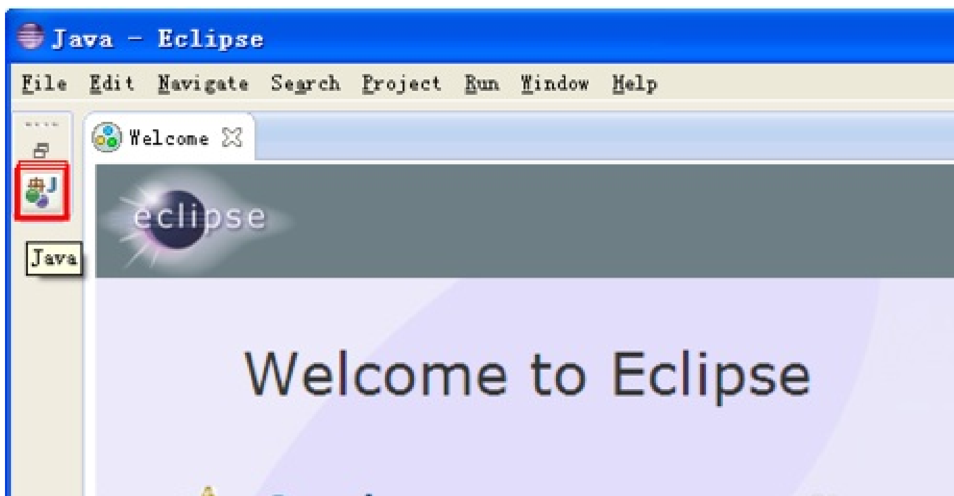

创建一个项目：选择file--New--java Project，如图：



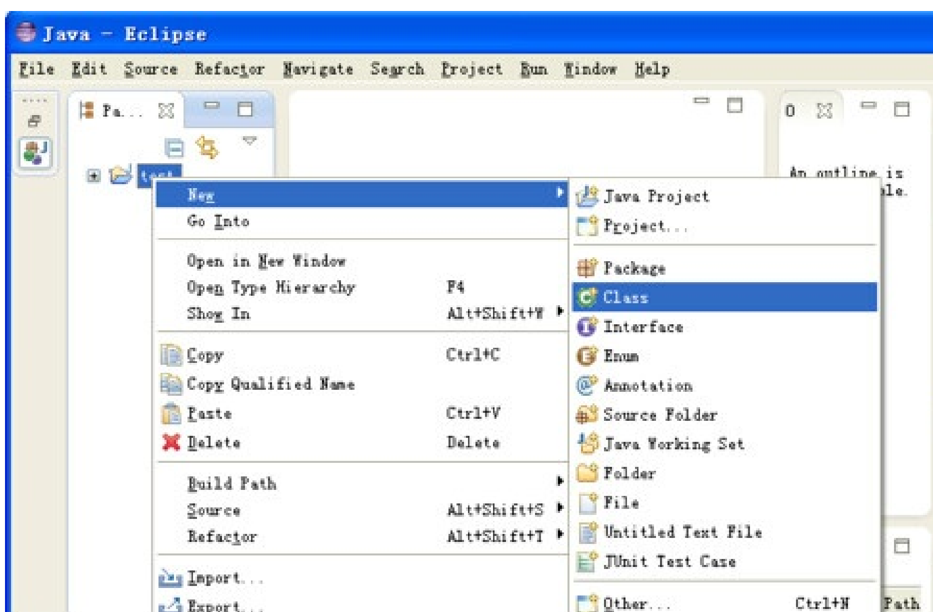

输入项目名称，比如我输入test，然后点击finish




完成项目的创建，点击红框里的小图标，如图：



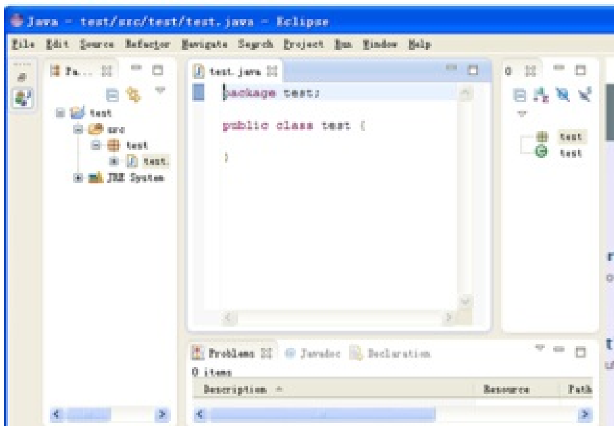

然后在左侧菜单选择test项目，右键--new--class




键入类名，如输入 test，如图，然后点击finish




这样在代码框里面你就可以开始输入代码啦！

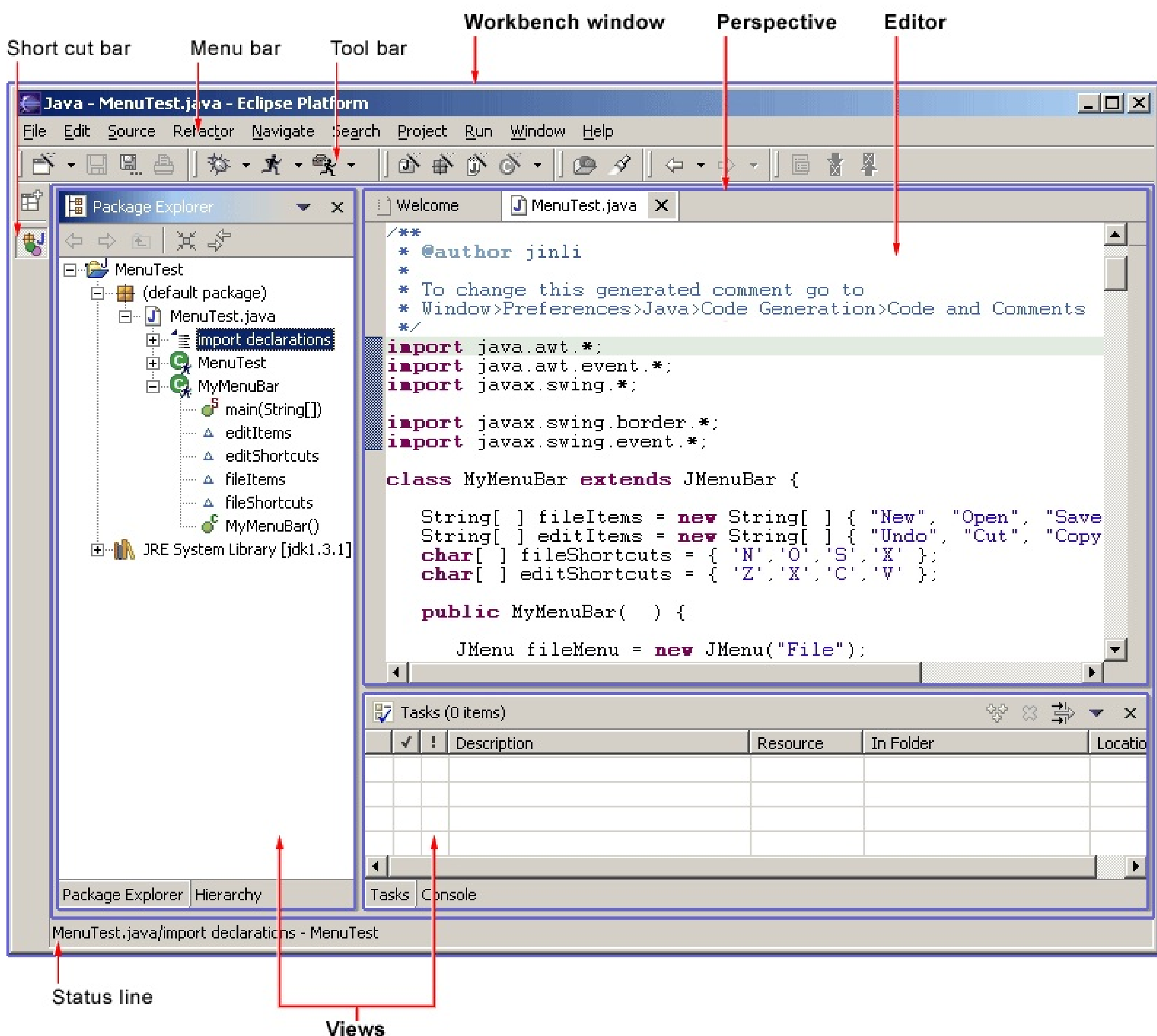


# Eclipse 窗口说明

## Eclipse 工作台(Workbench)

首先，让我们来看一下Eclipse 作台用户界面，和它里面的各种组件。

工作台是多个窗口的集合。每个窗口包含菜单栏，工具栏，快捷方式栏，以及一个或者多个透视图。




透视图是一个包含一系列视图和内容编辑器的可视容器。

视图完全存在于某个透视图中而且不能被共享，而任何打开的内容编辑器可以在透视图间共享。

如果两个或者多个透视图打开了同样的视图，他们共享这个视图的同一个实例，虽然在不同透视图之间视图的布局可能不同。

对于不同的工作台窗口中的透视图，编辑器和视图都不能共享。

一个透视图就好像是一本书里面的一页。它存在在一个窗口中，并且和其他透视图一起存在，和书中的一页一样，每次你只能看到一个透视图。

工作台的主菜单栏通常包括File，Edit，Navigate，Project，Window，Help这些顶层菜单。

其他的顶层菜单位于Edit和Project菜单之间，往往是和上下文相关，这个上下文包括当前活动的透视图，最前面的编辑器以及活动视图。

在File菜单中，你可以找到一个New子菜单，它包括Project，Folder，File的创建菜单项。

File 菜单也包含Import and Export菜单项，用来导入文件到Wrokbench中，以及导出它们。

在Edit菜单中，你可以找到象Cut，Copy，Paste，和Delete这些命令。这些命令称为全局命令，作用于活动部件。

也就是说，如果当Navigator活动时使用Delete命令, 实际操作是由Navigator完成的。

在Project菜单中，你可以找到和项目相关的命令，比如Open Project，Close Project和Rebuild Porject等。

在Run菜单中，你可以看到和运行，调试应用代码相关的命令，以及启动象Ant脚本这样的外部工具。

在Window菜单中，你可以找到Open Perspective子菜单，根据你开发任务的需要打开不同的透视图。

你也能看到透视图 布局管理菜单栏。Show View子菜单用来在当前的Workbench窗口中增加视图。

另外，你可以通过首选项菜单项来修改工作台的功能首选项配置。

如果你是插件开发者，你可以为平台提供新的视图，编辑器，向导，菜单和工具项。 这些东西都是用XML来定义的，插件一旦注册后，就可以和平台中已经存在的组件无缝地集成在一起。

# Eclipse 多窗口

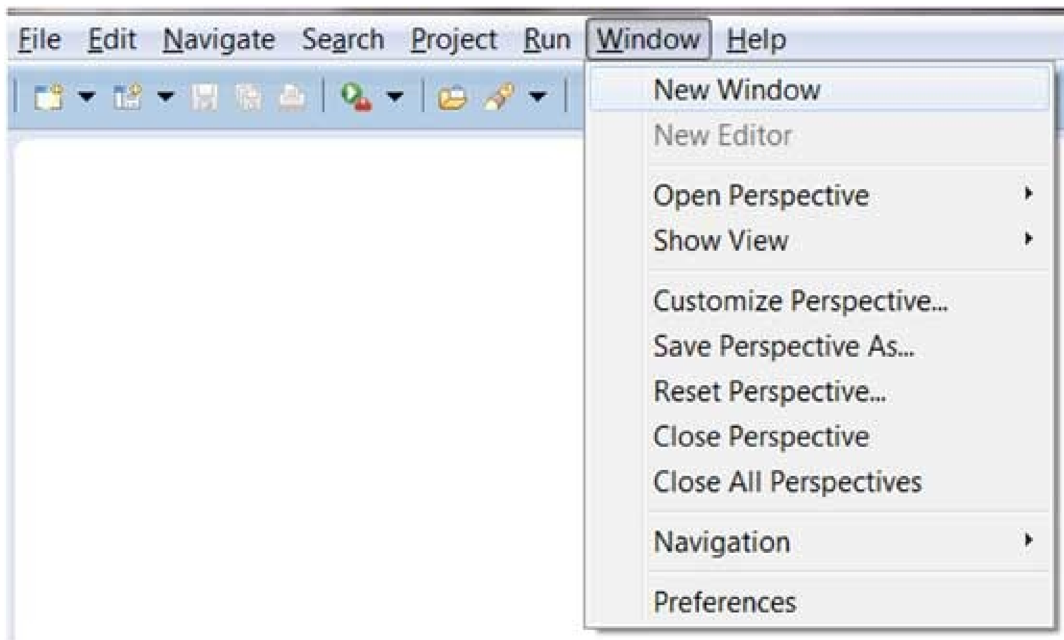
Eclipse 可以同时开启多个窗口，在 菜单栏选择： Window -> New Window 来开启多窗口。

多个窗口的切换你可以使用 Alt + Tab 来回切。

# Eclipse 菜单

Eclipse 查看的菜单栏通常包含以下几个菜单：

- File 菜单
- Edit 菜单
- Navigate 菜单
- Search 菜单
- Project 菜单
- Run 菜单
- Window 菜单
- Help 菜单




通过 Eclipse 插件你可以添加新的菜单和菜单项。

## 菜单描述

| 菜单名 | 描述 |
| --- | --- |
| File | File 菜单运行你打开文件，关闭编辑器，保存编辑的内容，重命名文件。 此外还可以导入和导出工作区的内容及关闭 Eclipse。 |
| Edit | Edit 菜单有复制和粘贴等功能。 |
| Source | 只有在打开 java 编辑器时 Source 菜单才可见。 Source 菜单关联了一些关于编辑 java 源码的操作。 |
| Navigate | Navigate 菜单包含了一些快速定位到资源的操作。 |
| Search | Search 菜单可以设置在指定工作区对指定字符的搜索。 |
| Project | Project 菜单关联了一些创建项目的操作。 |
| Run | Run 菜单包含了一些代码执行模式与调试模式的操作。 |
| Window | Window 菜单允许你同时打开多个窗口及关闭视图。 Eclipse 的参数设置也在该菜单下。 |
| Help | Help 菜单用于显示帮助窗口，包含了 Eclipse 描述信息，你也可以在该菜单下安装插件。 |

Eclipse 也可以自定义菜单，自定义菜单的详细介绍可以查看 Eclipse 透视图。

# Eclipse 视图

## 关于视图

Eclipse视图允许用户以图表形式更直观的查看项目的元数据。例如，项目导航视图中显示的文件夹和文件图形表示在另外一个编辑窗口中相关的项目和属性视图。

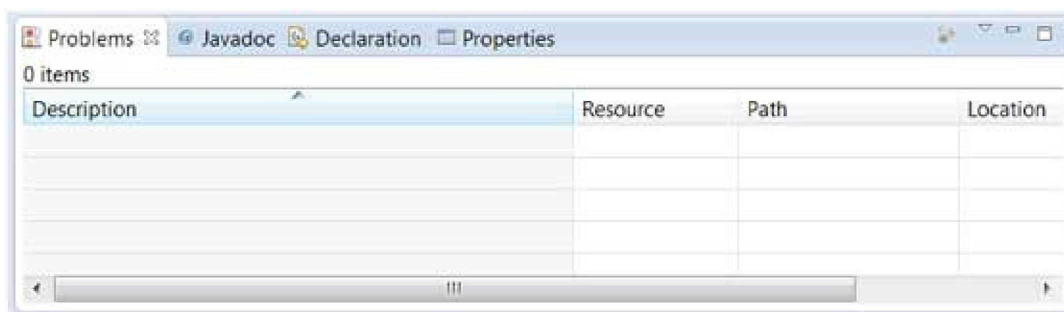
Eclipse 透视图(perspective) 可以显示任何的视图和编辑窗口。

所有的编辑器实例出现在一个编辑器区域内，可以通过文件夹视图查看。

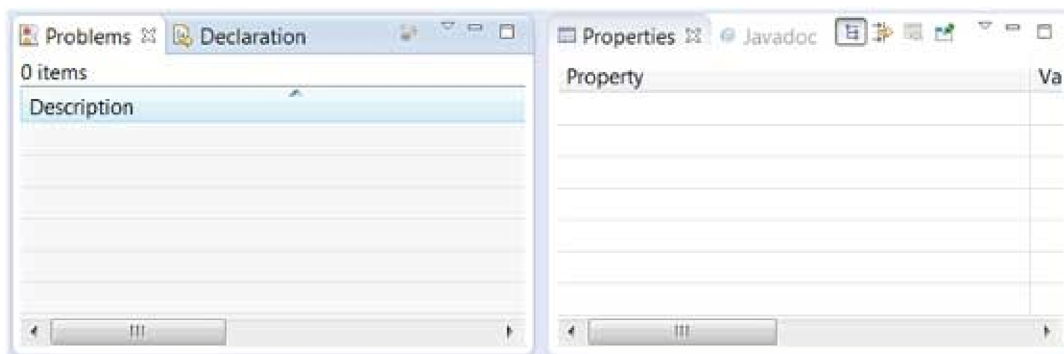
一个工作台窗口可以显示任意数量的文件夹视图。每个文件夹视图可以显示一个或多个视图。

## 组织视图

下图显示了文件夹视图的四个视图。

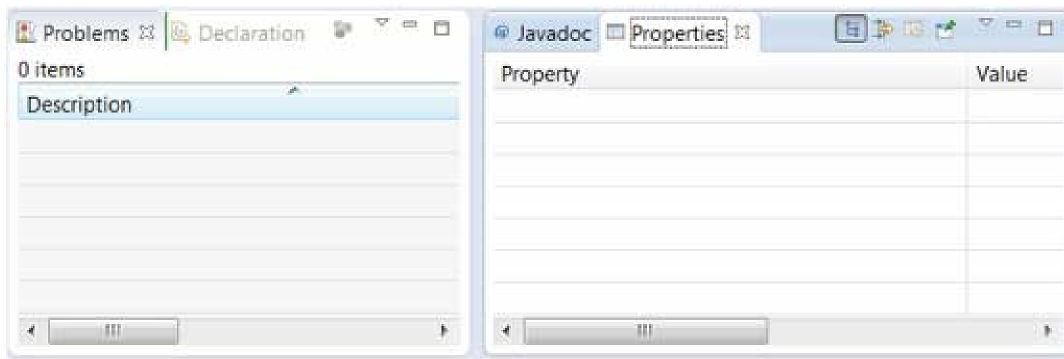



下图在两个文件夹视图中显示四个视图。



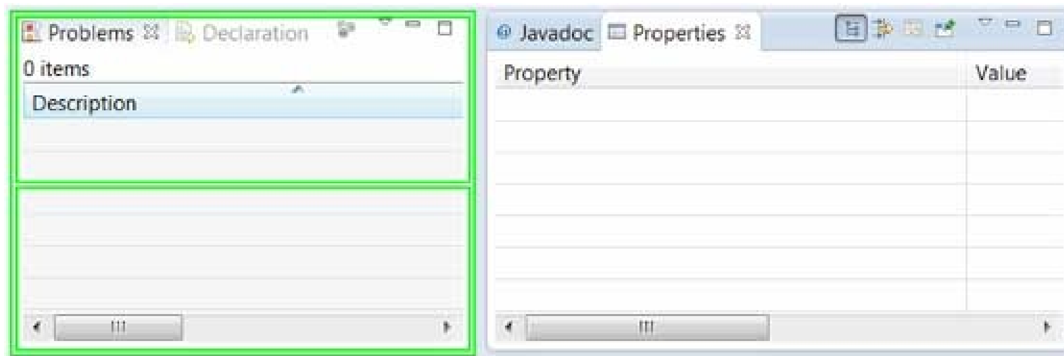

## 移除视图 views

视图从一个文件夹视图移动到另外一个文件夹视图只需要点击视图标题并推动视图工具区域到另外一个文件夹视图。




# 创建文件夹视图

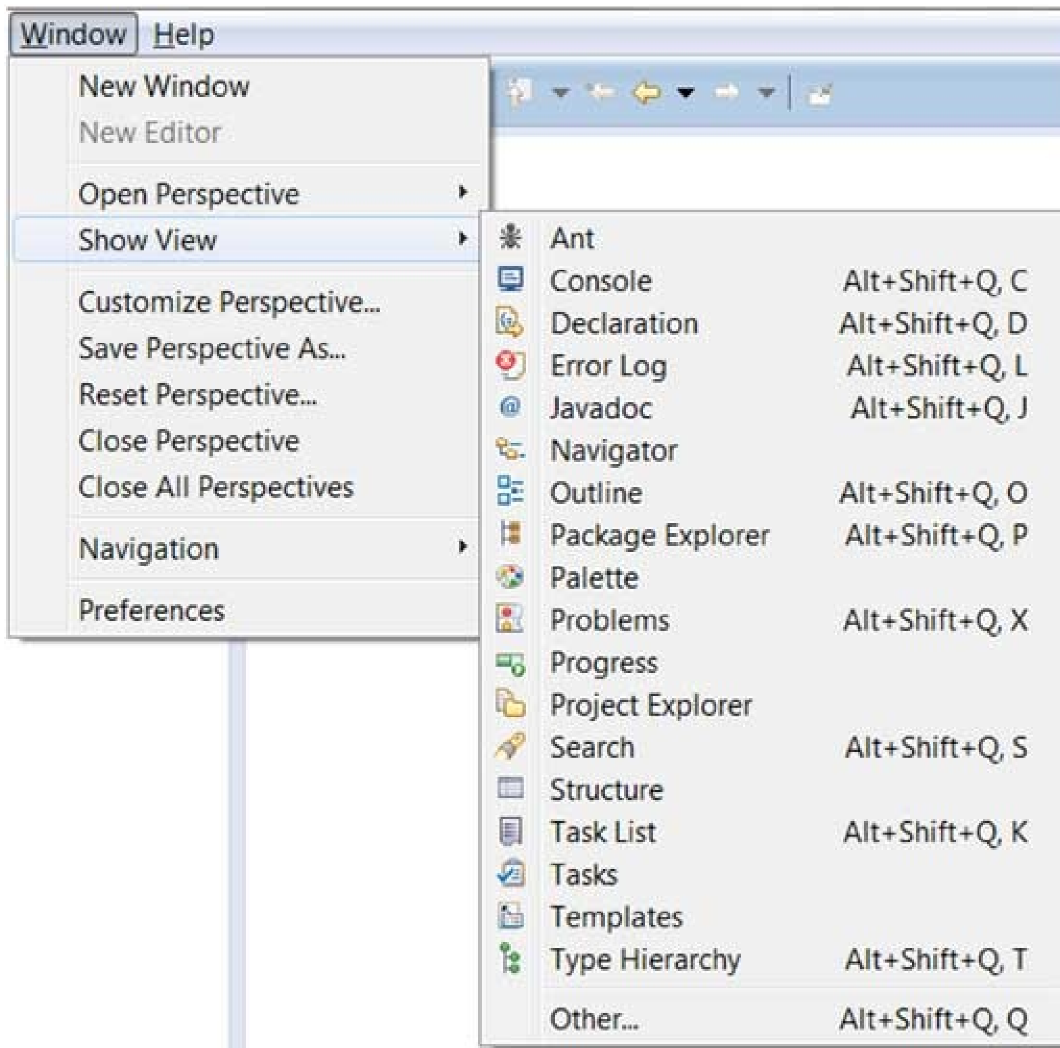
文件夹视图可以通过移动视图标题栏到编辑去外或移动标题栏到另外一个文件夹视图来动态创建。 下图中如果你拖动了绿色线框内的标题栏意味着一个新的文件夹视图将被创建。



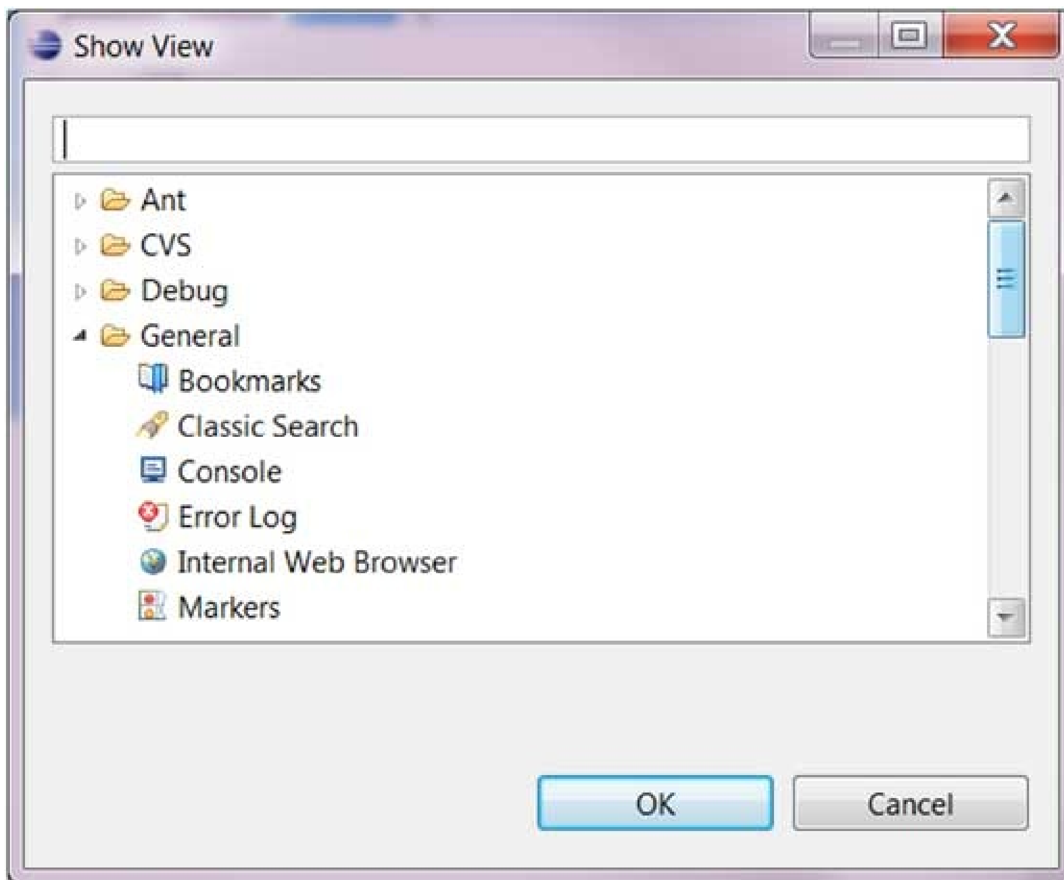

移动拖动图标到窗口的底部，您可以创建一个横跨窗口整个宽度的视图文件夹。移动拖动图标到窗口的左边或右边，您可以创建一个横跨窗口的整个高度视图文件夹。

# 操作视图

你可以在 Window 菜单中点击 "Show View" 选项打开其他视图。




点击 "Other" 菜单选项会弹出一个 "Show View" 对话框，对话框中你可以查找和激活视图。




视图通过各个分类来组织。你可以通过搜索框快速查找视图。 然后打开视图并选择，点击 "OK" 按钮即可。

## 什么是透视图？

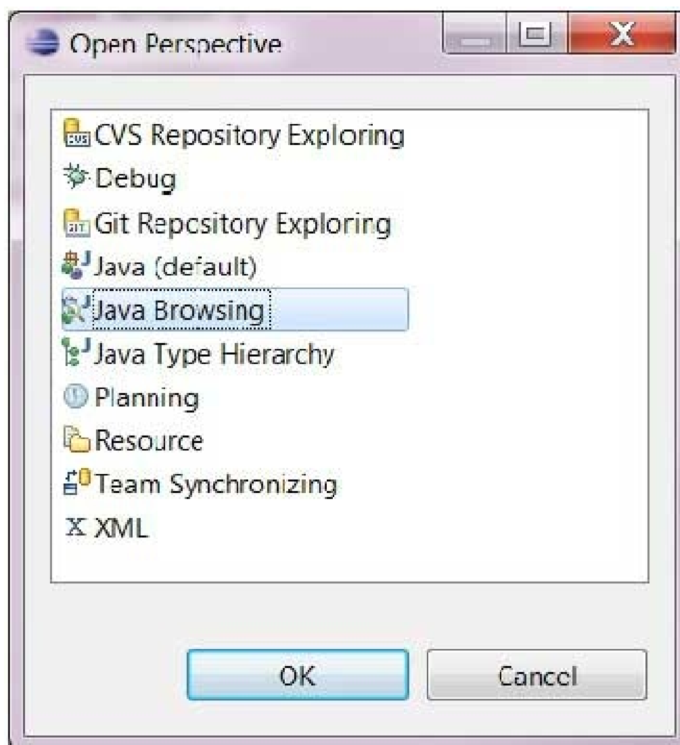
透视图是一个包含一系列视图和内容编辑器的可视容器。默认的透视图叫 java。

Eclipse 窗口可以打开多个透视图，但在同一时间只能有一个透视图处于激活状态。

用户可以在两个透视图之间切换。

## 操作透视图

通过"Window"菜单并选择"Open Perspective > Other"来打开透视图对话框。




透视图对话框中显示了可用的透视图列表。

该透视图列表也可以通过工具栏上的透视图按钮来打开 ( ) 。

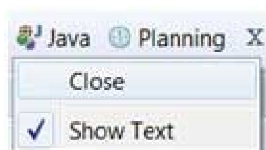
## 视图切换

大都数情况下 java 开发者会使用 Java 透视图和 Debug 透视图。你可以通过工具条上的透视图名称来自由切换。




## 关闭透视图

工具条上右击透视图名及选择"Close"项即可关闭透视图。

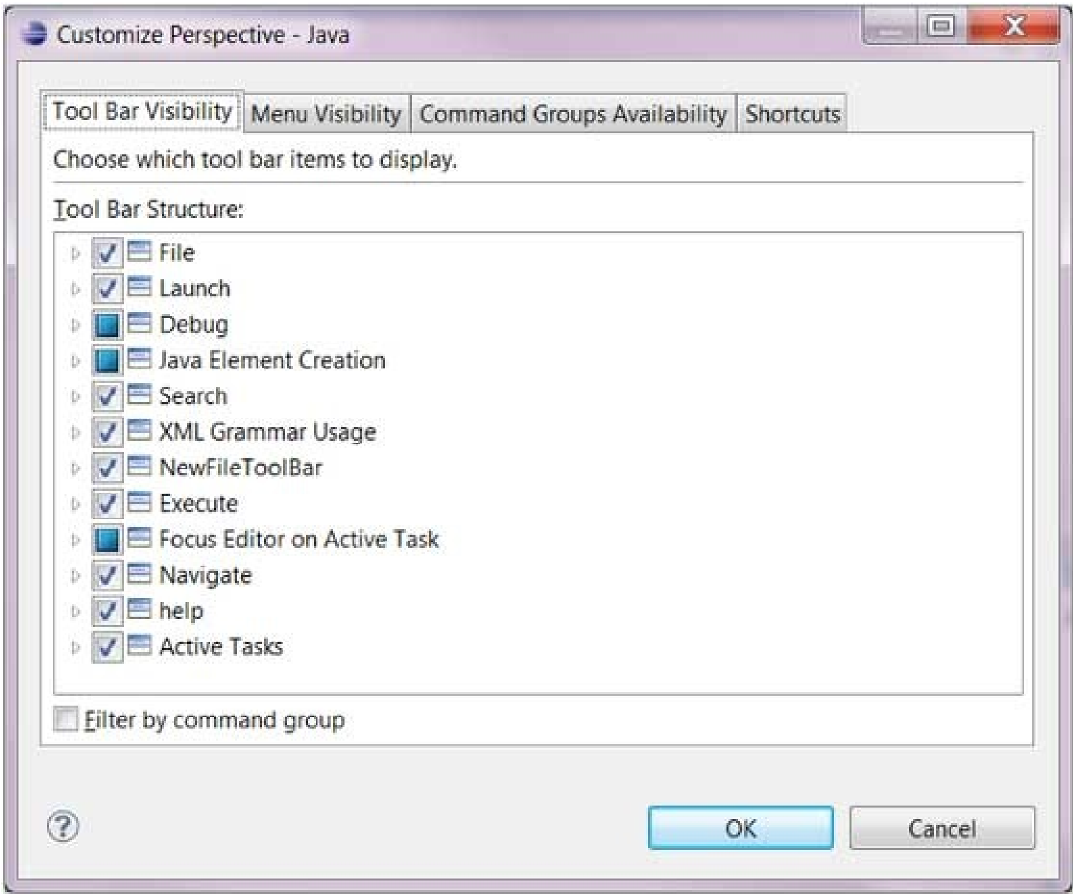


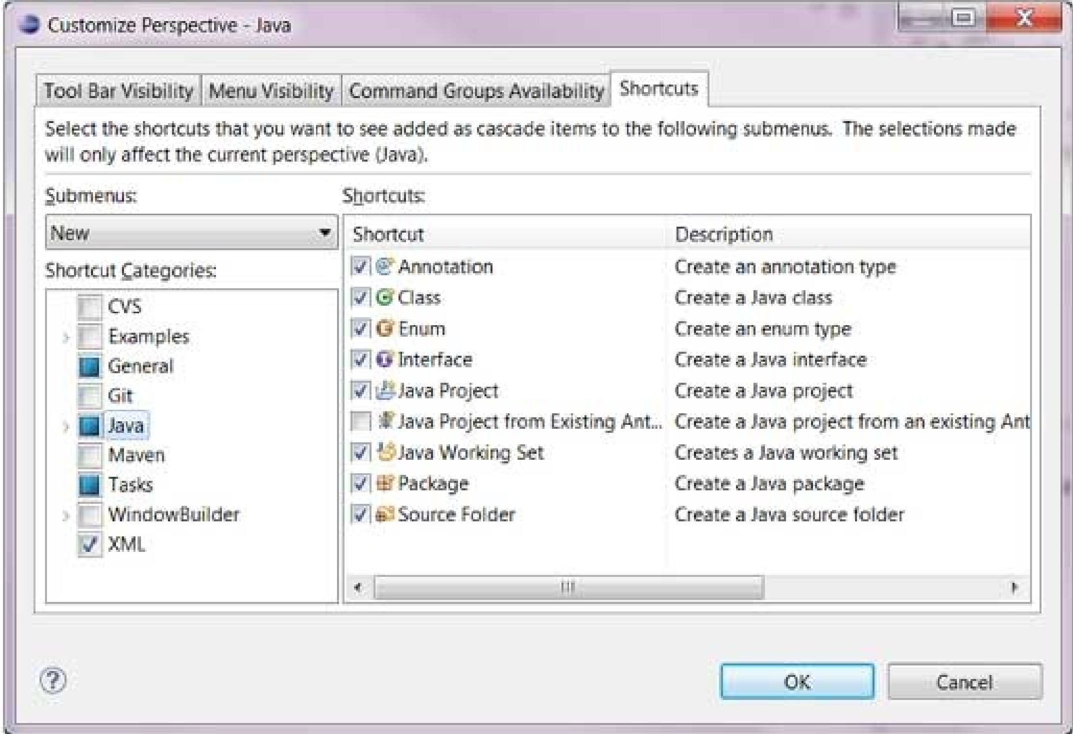

## 自定义透视图

我们可以通过自定义透视图窗口来设置我们想要的透视图。

- 点击菜单栏上的 "Windows" => "Customize Perspective" => 弹出窗口，可以在"Submenus"里面选择你要设置的内容。

- "New" => 设置你的新建菜单，可以把你平时需要新建的最常用的文件类型选中。
- "Show Views" 在你自定义的这个视图的布局，也就是切换到你自己的视图后，会出现哪些窗口。根据自己习惯进行设置。
- Open Perspective" => 切换视图菜单中，出现哪些可以选择的视图。
- 都设置好以后，保存你的自定义透视图。Windows => Save Perspective as，然后为你自定义的透视图取一个名字，保存。


Customize Perspective - Java
Tool Bar Visibility
Menu Visibility
Command Groups Availability
Shortcuts
Choose which tool bar items to display.
Tool Bar Structure:
File
Launch
Debug
Java Element Creation
Search
XML Grammar Usage
NewFileToolBar
Execute
Focus Editor on Active Task
Navigate
help
Active Tasks
Filter by command group
OK
Cancel

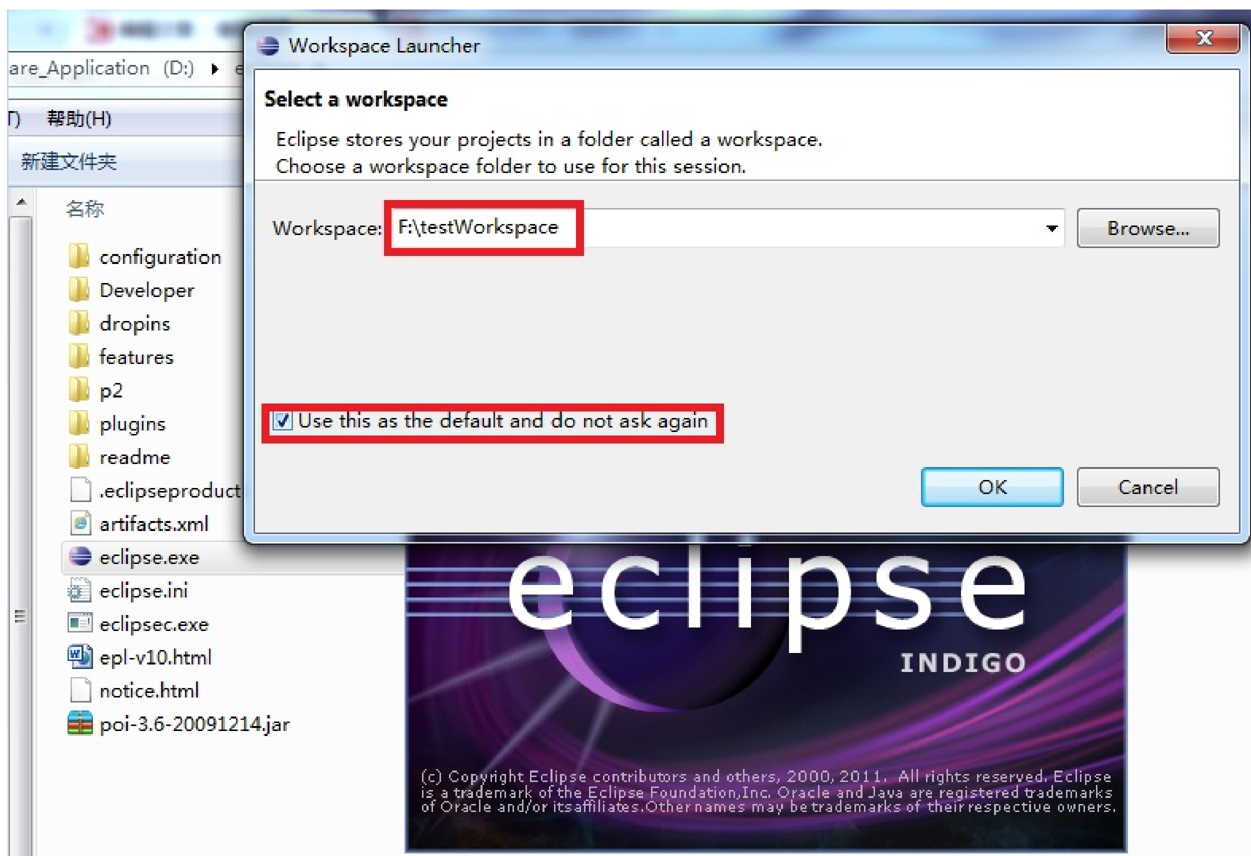

Customize Perspective - Java
Tool Bar Visibility
Menu Visibility
Command Groups Availability
Shortcuts
Select the shortcuts that you want to see added as cascade items to the following submenus. The selections made will only affect the current perspective (Java).
Submenus:
New
Shortcut Categories:
CVS
Examples
General
Git
Java
Maven
Tasks
WindowBuilder
XML
Shortcuts:
Shortcut
Description
Annotation
Create an annotation type
Class
Create a Java class
Enum
Create an enum type
Interface
Create a Java interface
Java Project
Create a Java project
Java Project from Existing Ant...
Create a Java project from an existing Ant
Java Working Set
Creates a Java working set
Package
Create a Java package
Source Folder
Create a Java source folder
OK
Cancel

# Eclipse 工作空间(Workspace)

eclipse 工作空间包含以下资源：

- 项目
- 文件
- 文件夹

项目启动时一般可以设置工作空间，你可以将其设置为默认工作空间，下次启动后无需再配置：

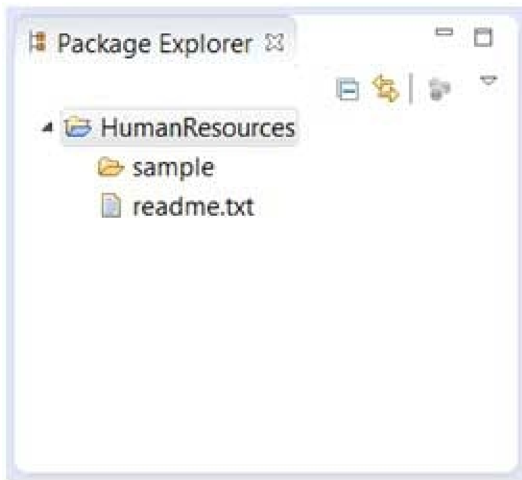



工作空间(Workspace)有明显的层次结构。 项目在最顶级，项目里头可以有文件和文件夹。

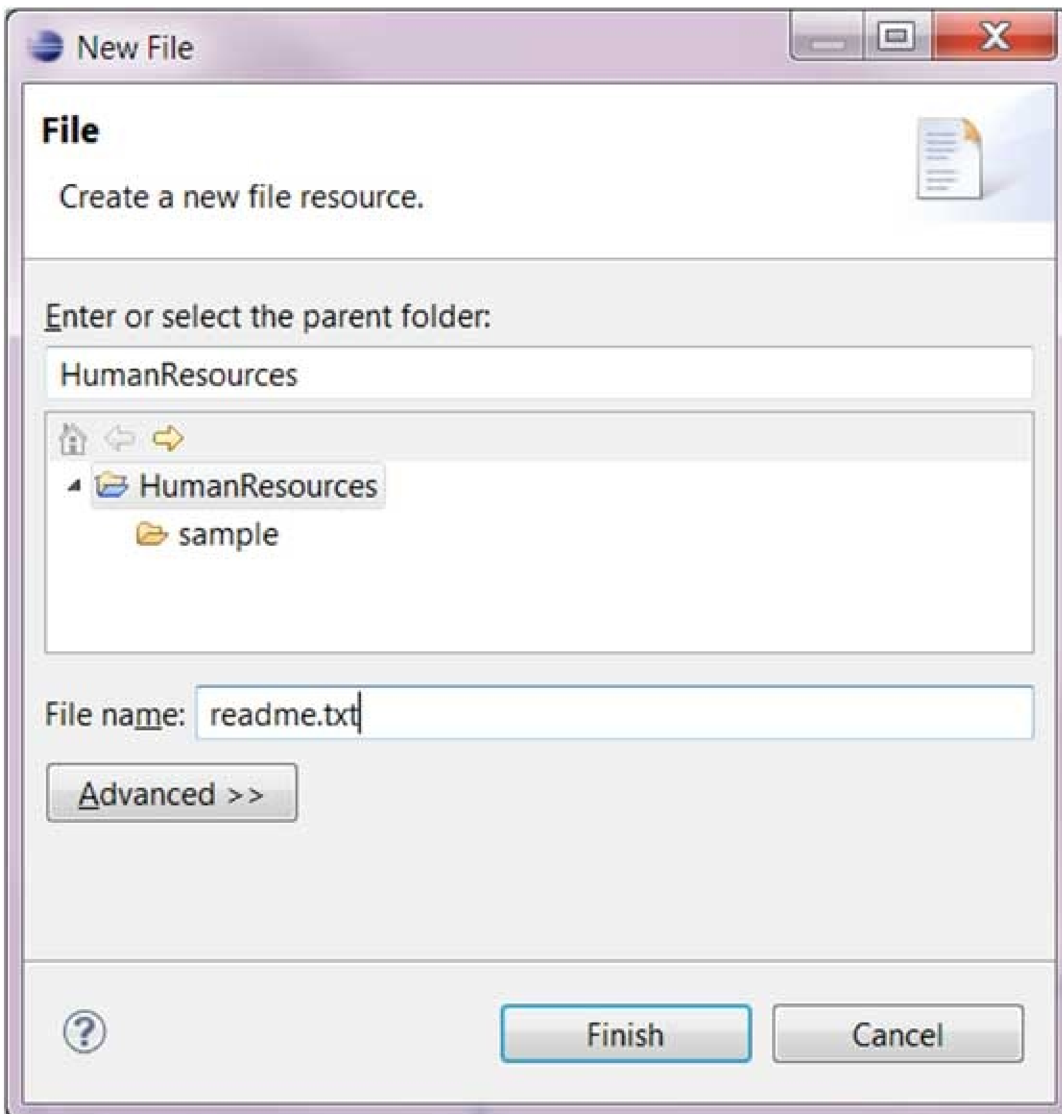
插件可以通过资源插件提供的API来管理工作空间的资源。

## 管理工作空间(Workspace)

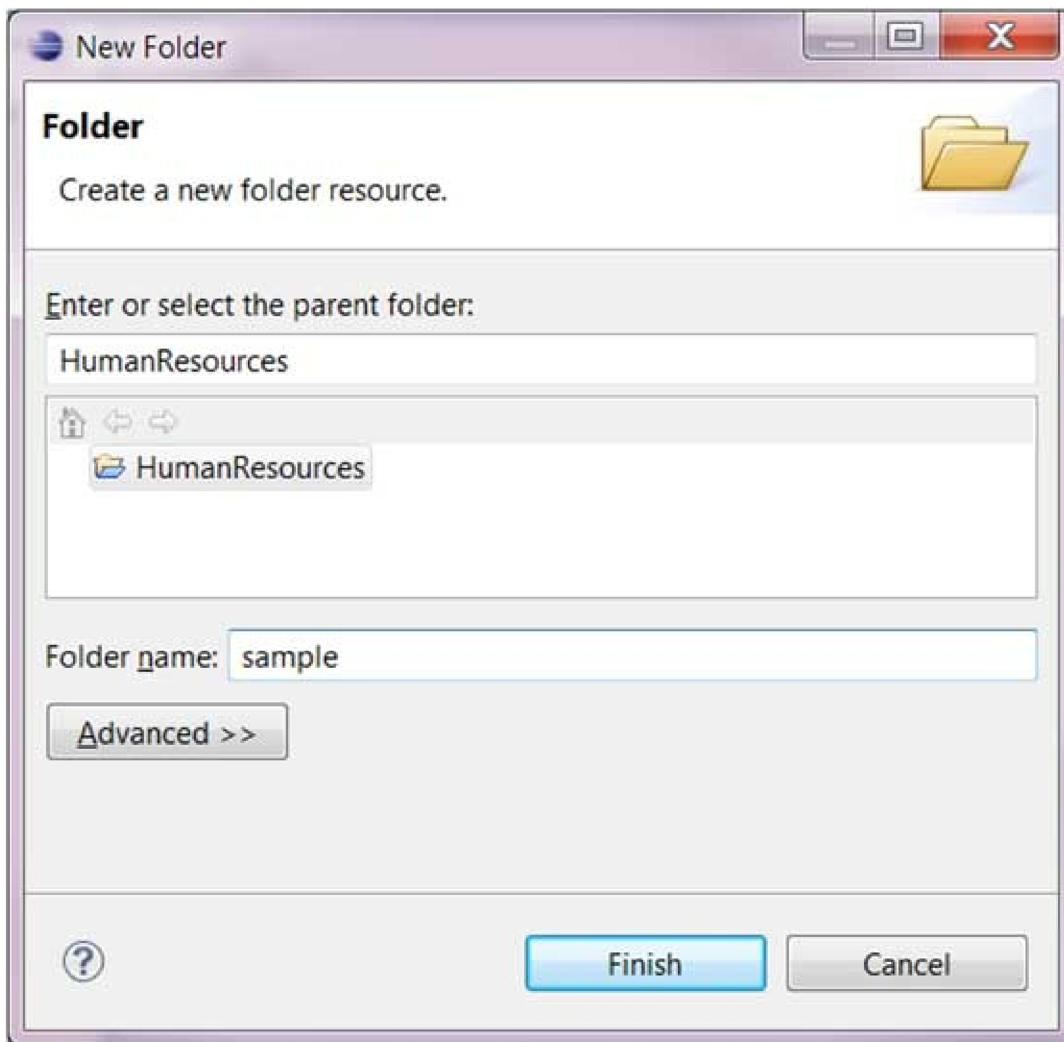
用户通过使用视图，编辑器和向导功能来创建和管理工作空间中的资源。其中，显示工作区的内容很多意见中的Project Explorer视图。显示项目工作空间内容的视图是Project Explorer视图。




文件创建向导(File > New > File)。

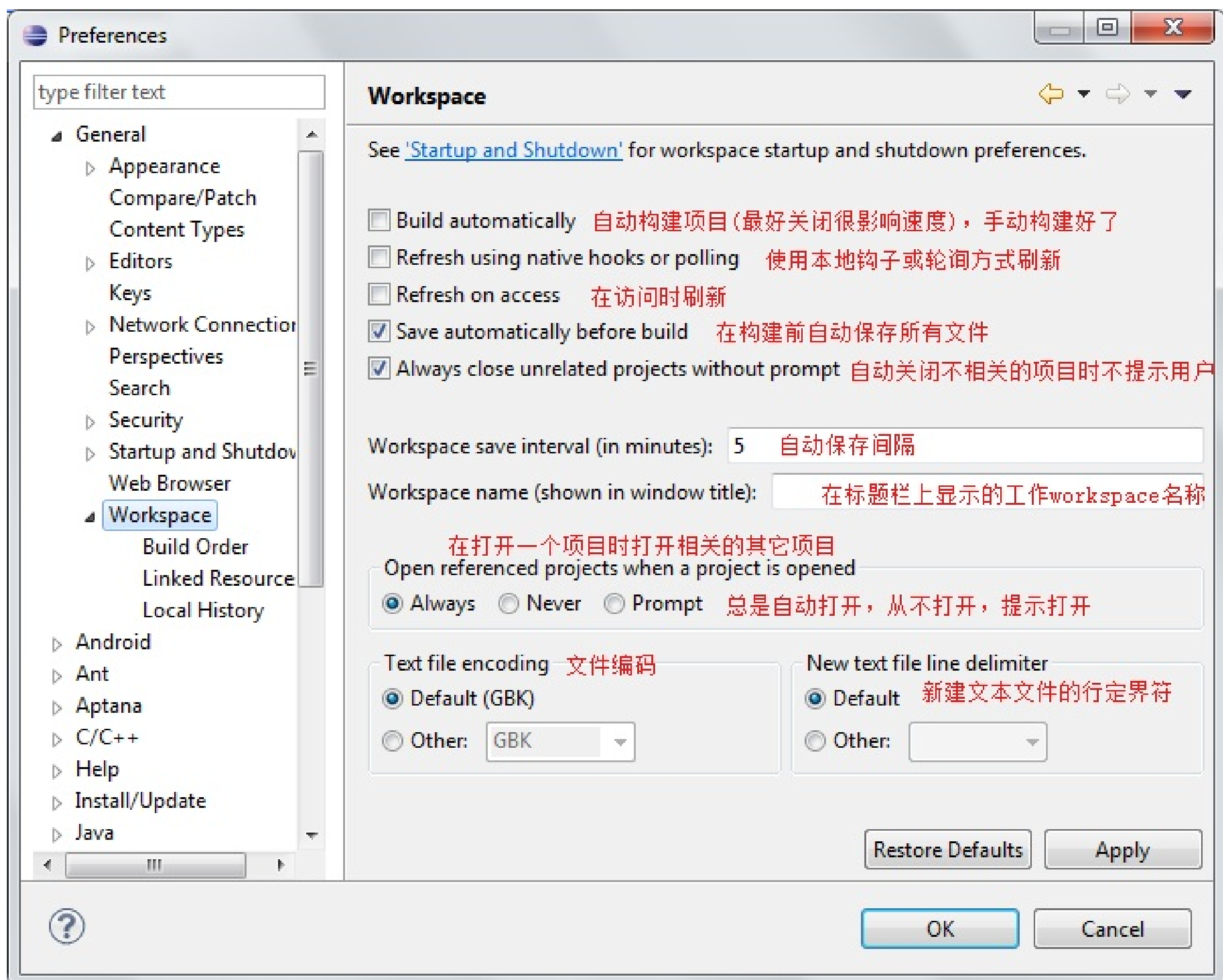



文件夹(Folder)创建向导(File > New > Folder)。



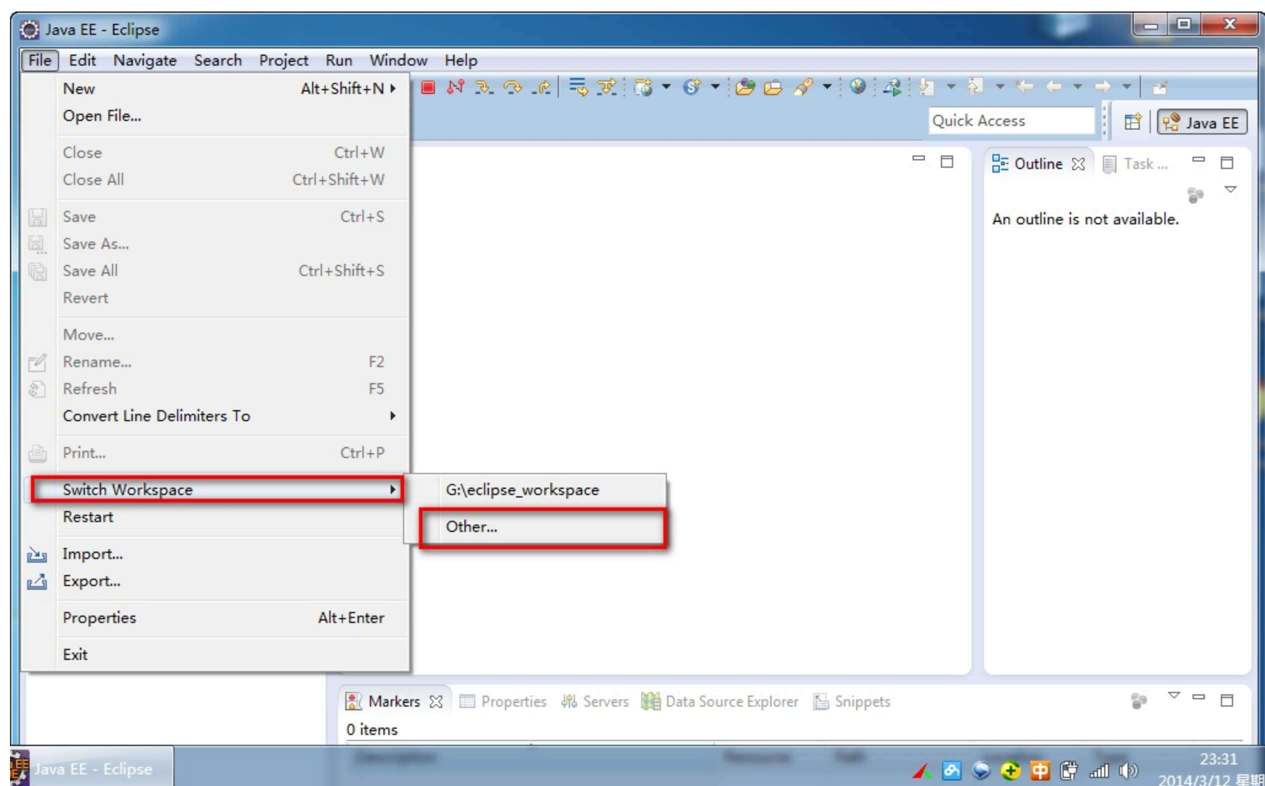

# 工作空间（workspace）设置

在菜单栏上选择 "Window" => "preferences..." => "General"=>"Workspace"，设置说明如下图：




## Eclipse切换工作空间(workspace)

Eclipse切换工作空间可以选择菜单栏中选择 "File" => "switch workspace" :


Java EE - Eclipse
File
Edit
Navigate
Search
Project
Run
Window
Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
G:\eclipse_workspace
Other...
Quick Access
Java EE
Outline
Task ...
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 items
Java EE - Eclipse
23:31
2014/3/12

# Eclipse 创建 Java 项目

## 打开新建 Java 项目向导

通过新建 Java 项目向导可以很容易的创建 Java 项目。打开向导的途径有：

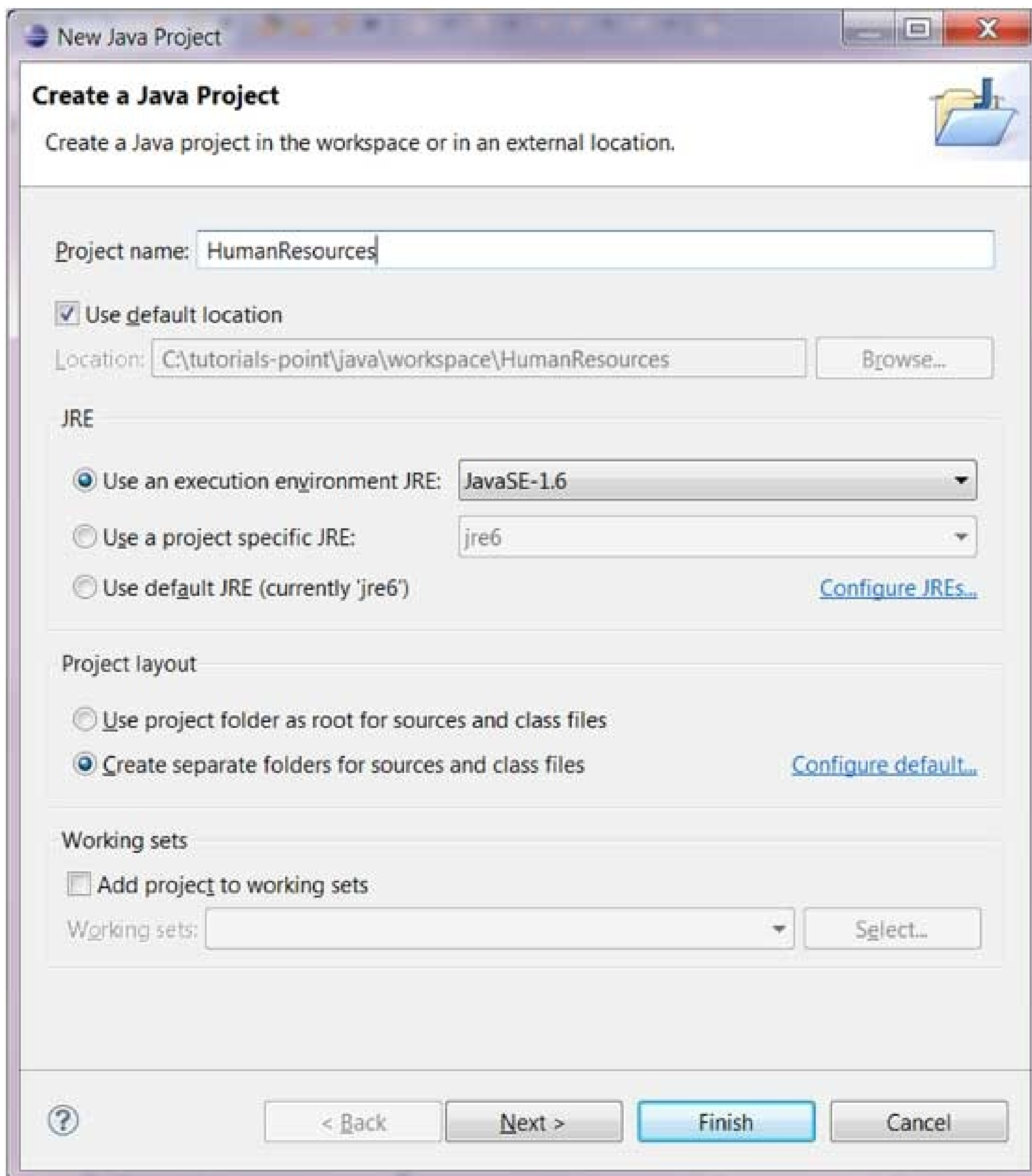
- 通过点击 "File" 菜单然后选择 New > Java Project
- 在项目浏览器(Project Explorer)窗口中鼠标右击任一地方选择 New > Java Project
- 在工具条上点击新建按钮 ( ) 并选择 Java Project

## 使用新建 Java 项目向导

新建 Java 项目向导有两个页面。

第一个页面:

- 输入项目名称（Project Name 栏中）
- 选择 Java Runtime Environment (JRE) 或直接采用默认的
- 选择项目布局（Project Layout），项目布局决定了源代码和 class 文件是否放置在独立的文件夹中。 推荐的选项是为源代码和 class 文件创建独立的文件夹。

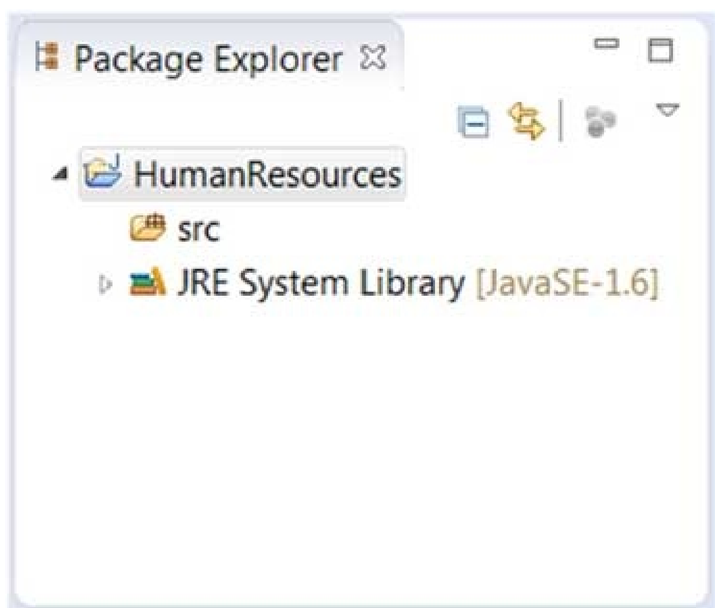



你可以点击"Finish" 按钮来创建项目或点击"Next" 按钮来修改 java 构建的配置。

第二个页面 Java 构建路径设置（Java Build Settings ）， 该页面我们可以配置项目的依赖关系及额外的 jar 包。

# 查看新建项目

Package Explorer 显示了新建的 Java 项目。项目图标中的 "J" 字母表示 Java 项目。 文件夹图标表示这是一个 java 资源文件夹。


Package Explorer
HumanResources
src
JRE System Library [JavaSE-1.6]

# Eclipse 创建 Java 包

## 打开新建 Java 包向导

你可以使用新建 Java 包向导来创建 Java 包。Java 包向导打开方式有：

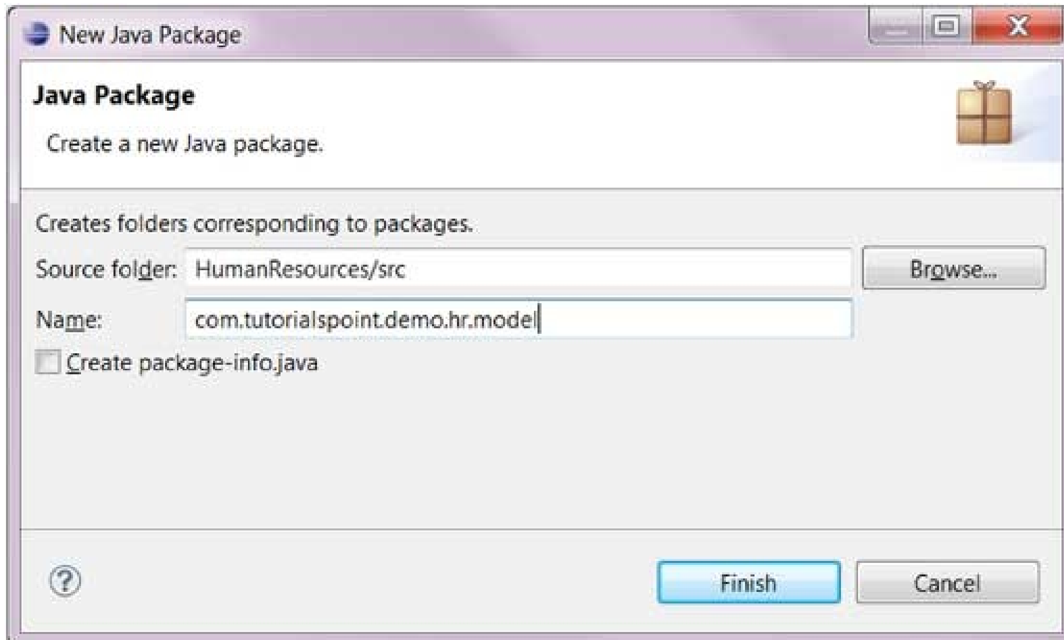
- 通过点击 "File" 菜单并选择 New > Package
- 在 Package Explorer 中通过右击鼠标选择 > Package
- 在工具条上点击包按钮( )

如果你要创建子包，在打开创建 Java 包向导前选择好父包，这样在名称字段就有了父包的值。

## 使用创建 Java 包向导

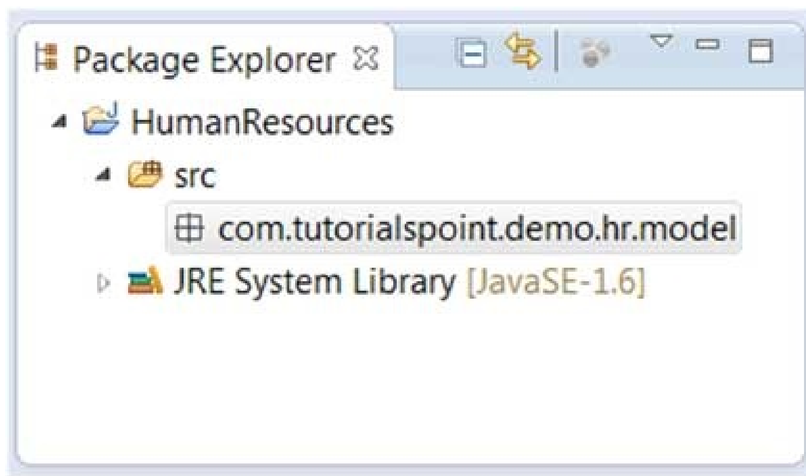
在向导弹出框(New Java Package)中可以执行以下操作：

- 输入资源文件夹名(Source Folder 字段)
- 输入包名(Name 字段)
- 点击 "Finish"按钮




## 查看新建包

在 Package Explorer 的资源文件夹下我们可以查看到新建的包。


Package Explorer
HumanResources
src
com.tutorialspoint.demo.hr.model
JRE System Library [JavaSE-1.6]

# Eclipse 创建 Java 类

## 打开新建 Java 类向导

你可以使用新建 Java 类向导来创建 Java 类，可以通过以下途径打开 Java 类向导：

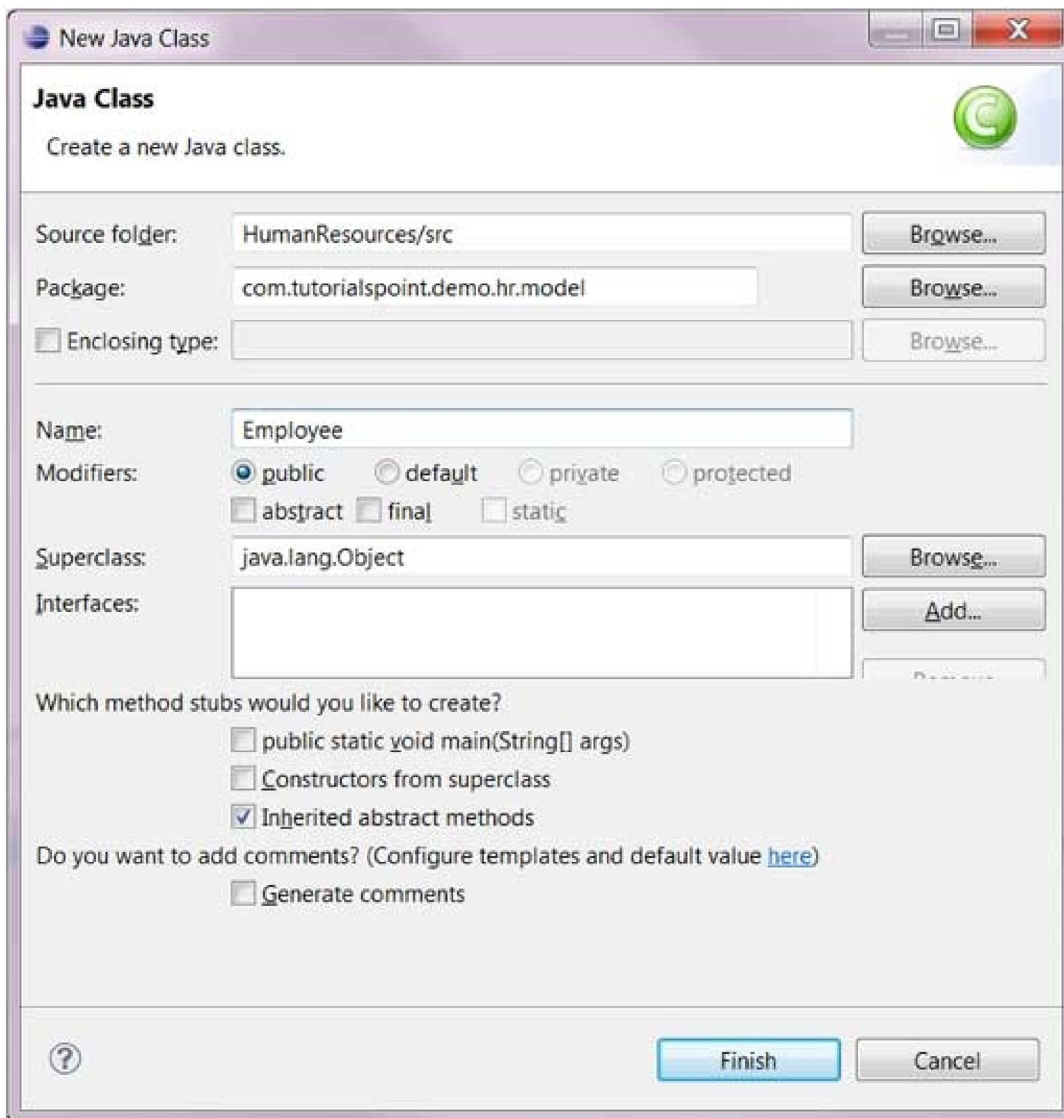
- 点击 "File" 菜单并选择 New > Class
- 在 Package Explorer 窗口中右击鼠标并选择 New > Class
- 点击类的下拉按钮 ( ) 并选择 ( )

在打开创建 Java 类向导前，最好选择好Java类所属的包名，这样在创建 Java 类时包名字段就会自动填充。

## 使用新建 Java 类向导

Java 类向导的弹窗中你可以进行以下操作：

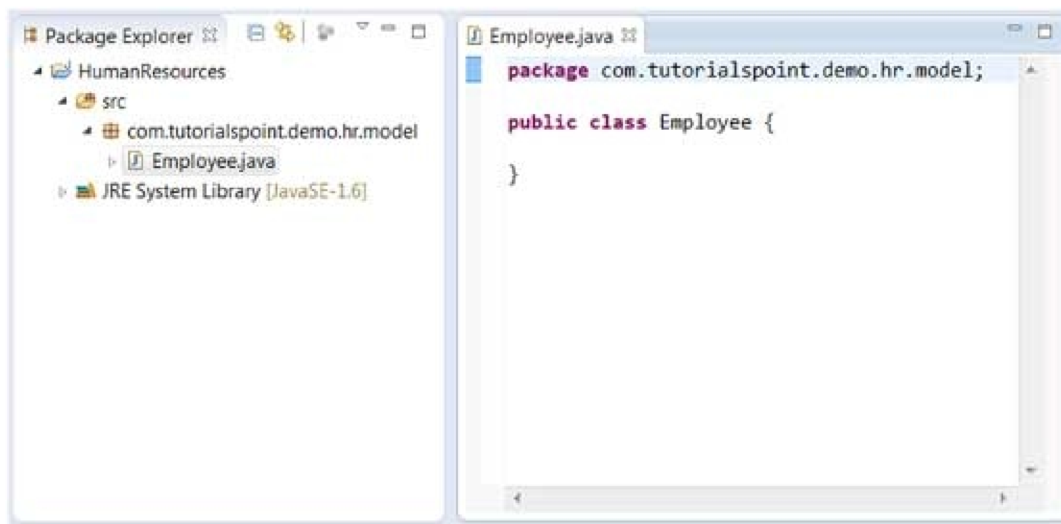
- 确认文件夹名(Source Folder)和包名(Package)是否正确
- 输入类名
- 选取其他修饰类
- 输入超类（Superclass）的名称或点击 Browse(浏览)按钮选择已存在的类
- 点击 Add(添加) 按钮选择类实现的接口
- 在复选框中可以选择方法创建方式及是否自动生成注释




- 点击 Finish(完成)按钮

# 查看新建的 Java 类

在 Package Explorer 视图中我们可以看到新建的类，我们可以通过右边的Java编辑器修改代码。


Package Explorer
HumanResources
src
com.tutorialspoint.demo.hr.model
Employee.java
JRE System Library [JavaSE-1.6]
Employee.java
package com.tutorialspoint.demo.hr.model;
public class Employee {
}

# Eclipse 创建 Java 接口

## 打开新建 Java 接口向导

新建 Java 接口向导可以创建新的 Java 接口。打开向导的方式有：

- 点击 File 菜单并选择 New > Interface
- 在 Package Explorer 窗口中右击鼠标并选择 New > Interface
- 在工具条上的下拉框按钮中 ( ) 选择 ( )

在打开创建 Java 接口向导前，最好选择好Java接口所属的包名，这样在创建 Java 接口时包名字段就会自动填充。

## 使用新建 Java 接口向导

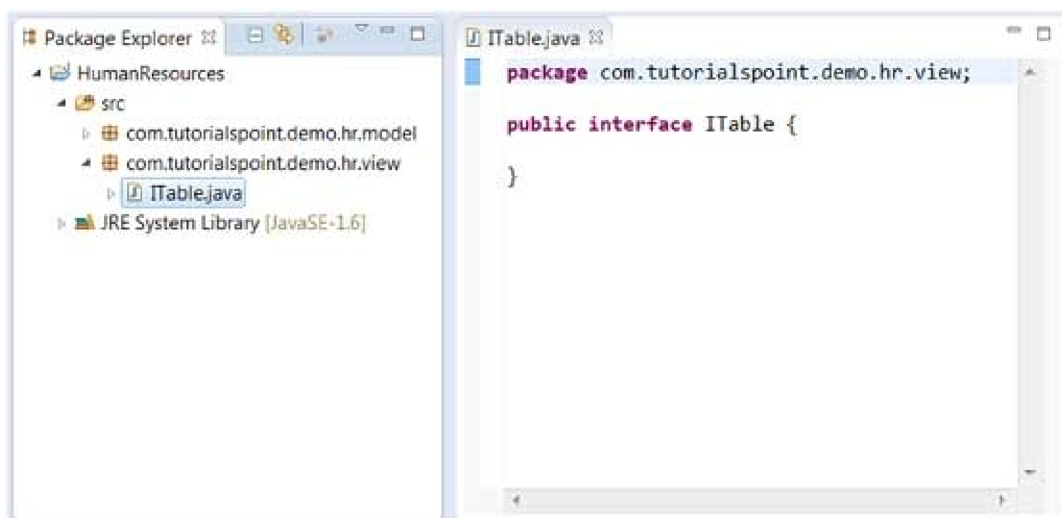
Java 接口向导的弹窗中你可以进行以下操作：

- 确认文件夹名(Source Folder)和包名(Package)是否正确
- 输入接口名称
- 点击 Add(添加) 按钮并选择要接口，该接口将被继承
- 选择是否自动生成注释
- 点击 Finish(完成) 按钮



## 查看新建的 java 接口

在 Package Explorer 视图中我们可以看到新建的接口，我们可以通过右边的Java编辑器修改接口代码。



# Eclipse 创建 XML 文件

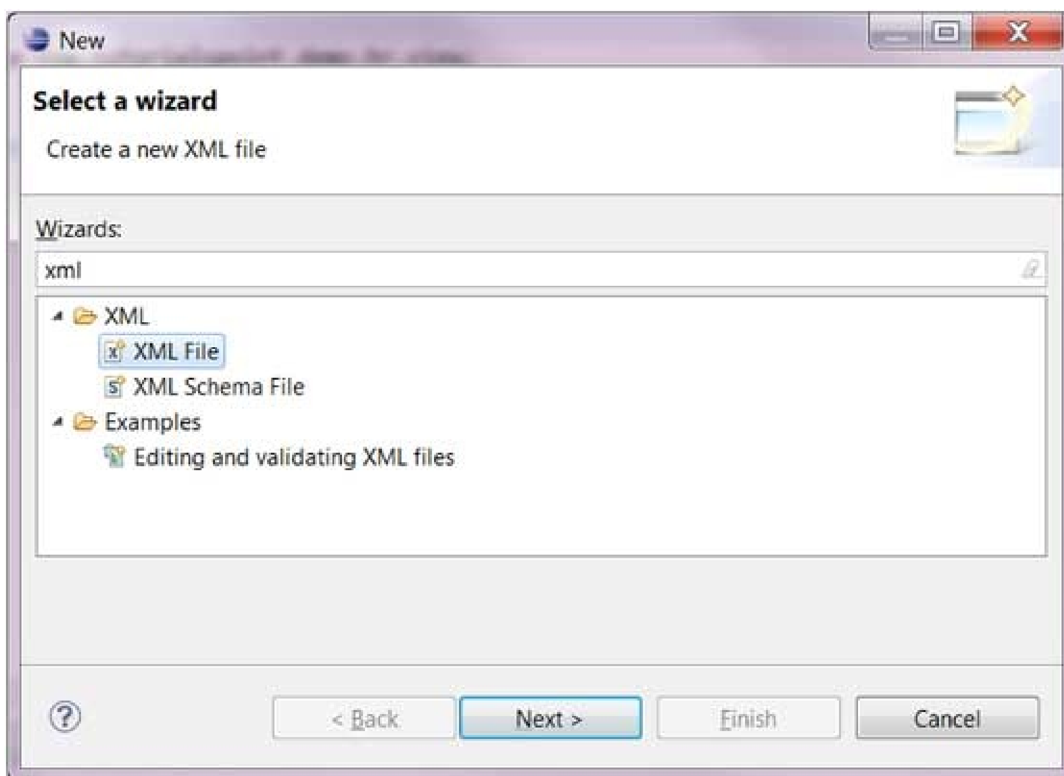
## 打开新建 XML 文件向导

你可以使用新建 XML 文件向导来创建 XML 文件。打开向导的方式有：

- 点击 File 菜单并选择 New > Other
- 点击新建下拉框 ( ) 选择 Other
- 快捷键组合： ctrl + N

在向导对话框中可以进行以下操作：

- 在输入框中输入 XML，会显示关联 XML 的向导
- 在展开的 XML 类别中选择 XML 文件




- 点击 Next 按钮进入新建 XML 文件向导

### 注意:

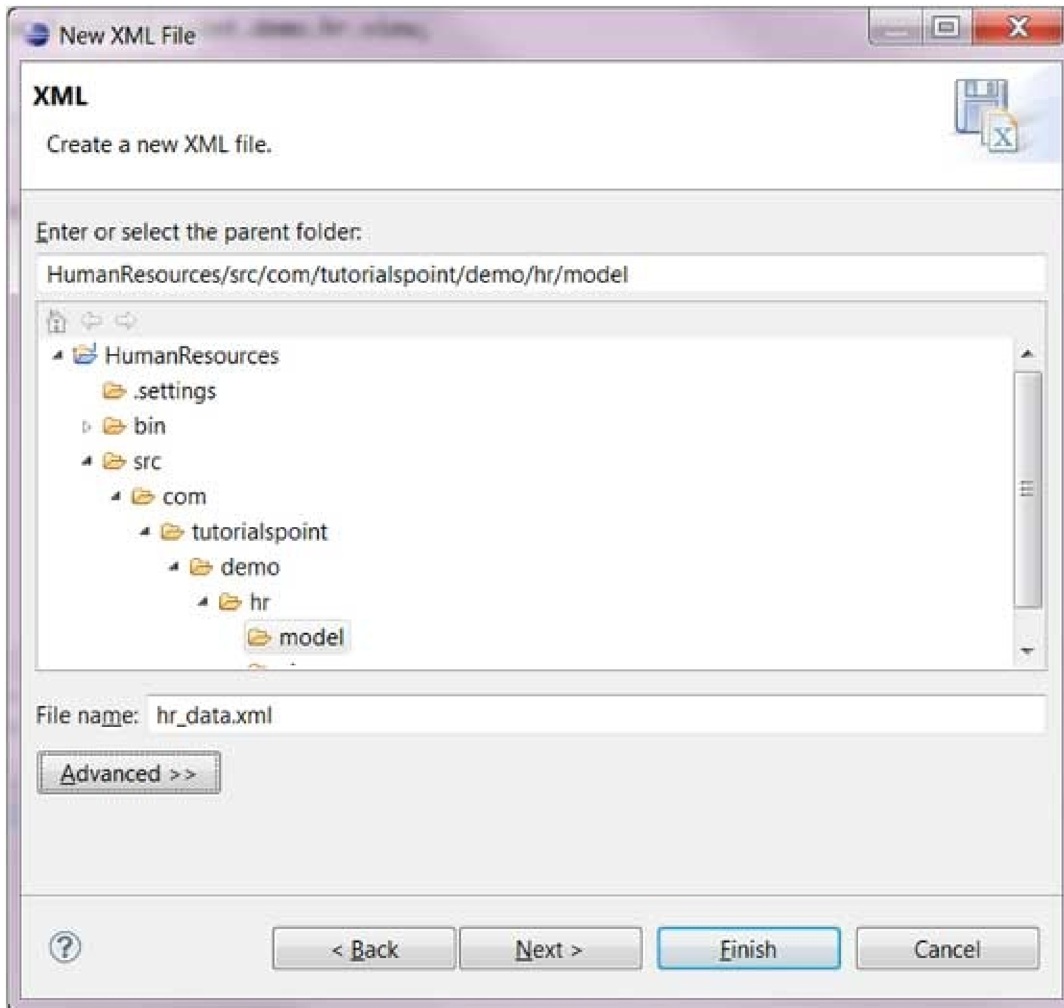
接下来在向导中我们还可以做以下操作：

- 点击 File 菜单并选择 New > XML File
- 在工具条上点击 XML File 按钮 ( XML File... )

# 使用新建的 XML 文件向导

在新建 XML 文件向导中我们可以进行如下操作：

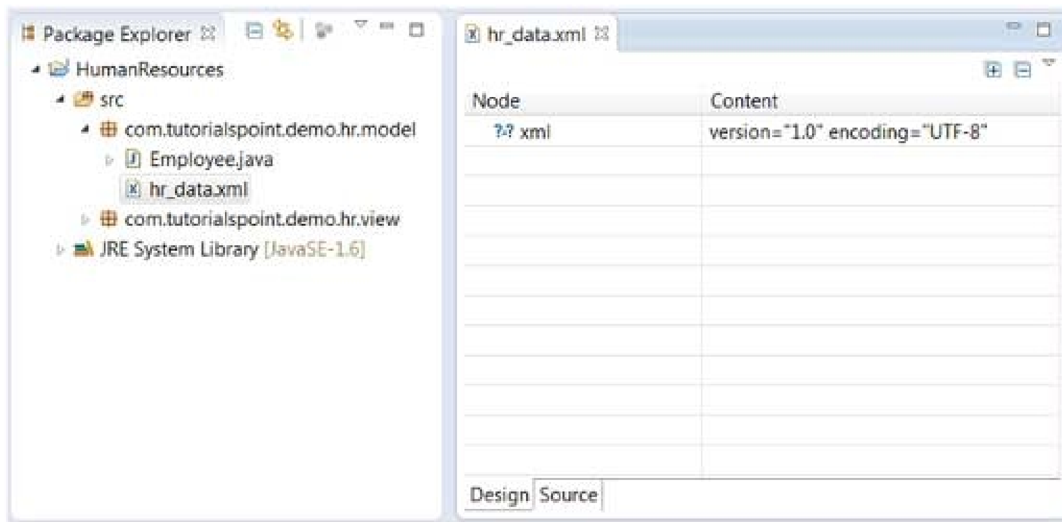
- 输入或选择 XML 文件所属的文件夹
- 输入 xml 文件名




- 点击 Next 按钮可以配置 DTD, XML Schema 的 XML 模式描述语言， 或者你可以直接点击 Finish 按钮完成 XML 文件的创建。

# 查看新建的 XML 文件

在 Package Explorer 视图中我们可以看到新建的 XML 文件，在右边的 XML 编辑器中我们可以修改新建的 XML 文件 。




XML 编辑器可以使用视图模式或源码模式来设计 XML 文件。

# Eclipse Java 构建路径

## 设置 Java 构建路径

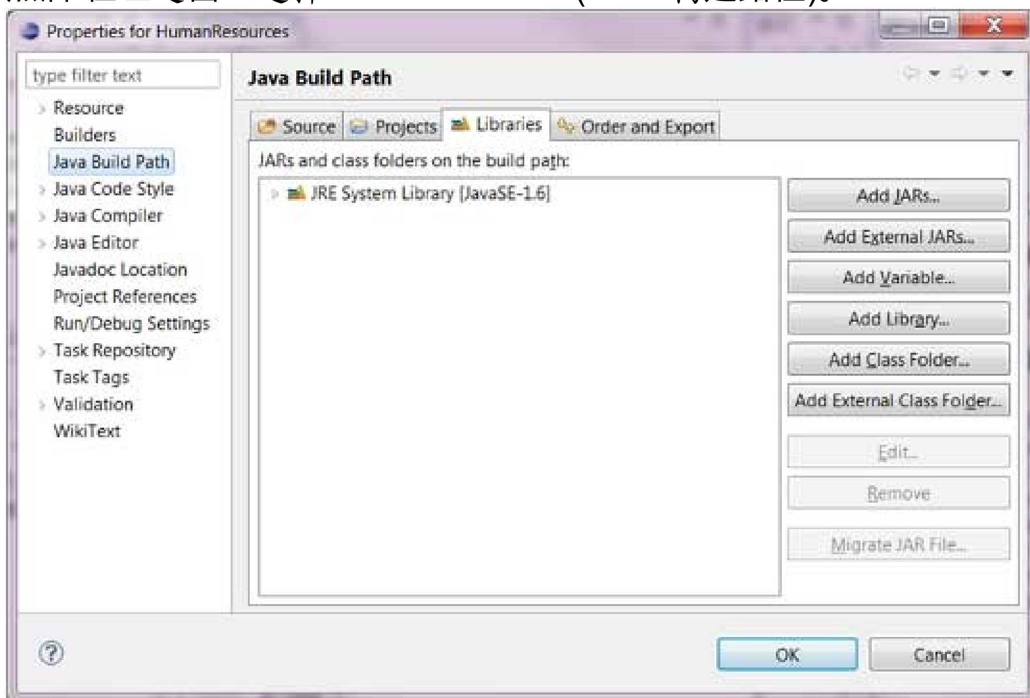
Java构建路径用于在编译Java项目时找到依赖的类，包括以下几项：

- 源码包
- 项目相关的 jar 包及类文件
- 项目引用的的类库

我们可以通过使用 Java 项目属性对话框中的 Java Build Path(Java 构建路径)选项来查看和修改 Java 构建路径。

Java 项目属性对话框可以通过在 Package Explorer 视图中鼠标右击指定的 Java 项目并选择 Properties(属性) 菜单项来调用。

然后 在左边窗口选择 Java Build Path(Java 构建路径)。




在 Java 构建路径窗口中我们可以已经引用到的 jar 包。

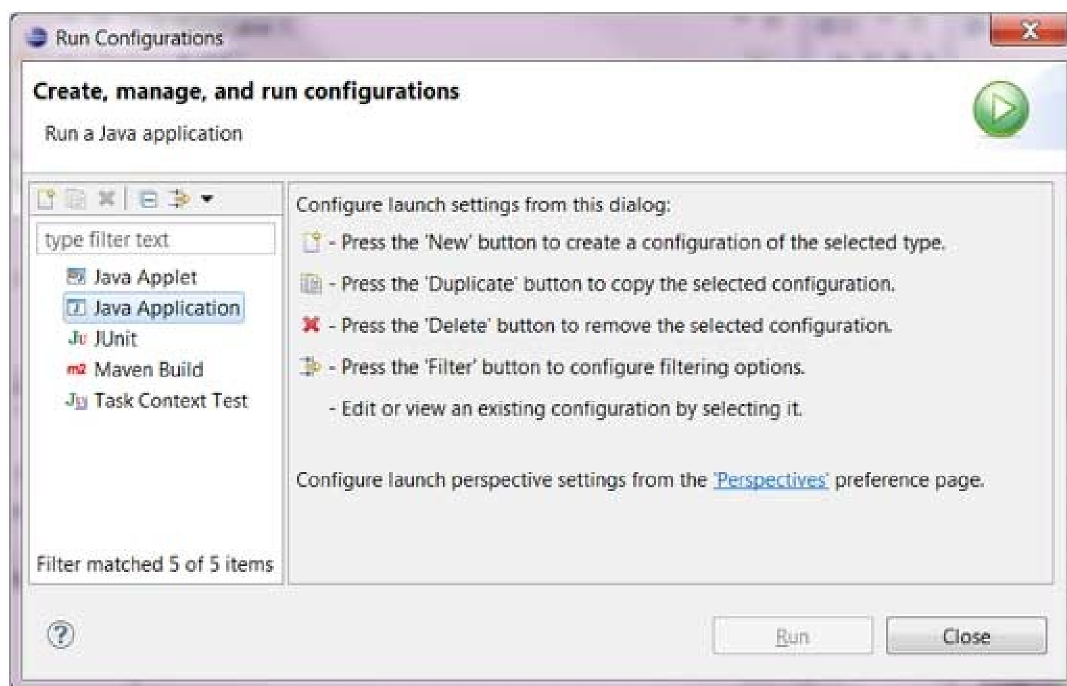
引用 jar 包可以在 Libraries 选项卡中完成，在 Libraries 选项卡中我们可以通过点击 Add JARs 来添加 Eclipse 工作空间中存在的jar包或 点击 External JARs 来引入其他文件中的 jar 包。

# Eclipse 运行配置(Run Configuration)

## 创建和使用 Eclipse 运行配置

在运行配置(Run Configuration)对话框中可以创建多个运行配置。每个配置可以在应用中启用。

运行配置(Run Configuration)对话框可以通过 Run 菜单中选择 Run Configurations 来调用。



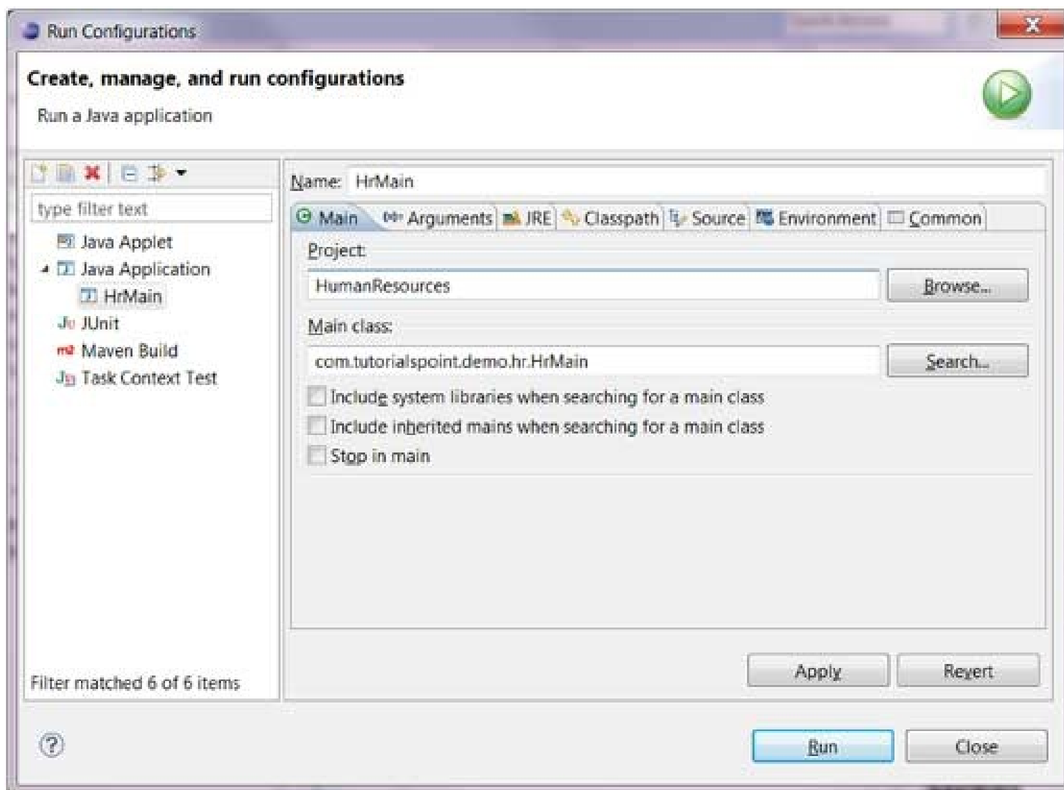

如果要给 Java 应用创建运行配置需要在左侧列表中选择 "Java Application" 并点击 New 按钮。

对话框中描述的项有：

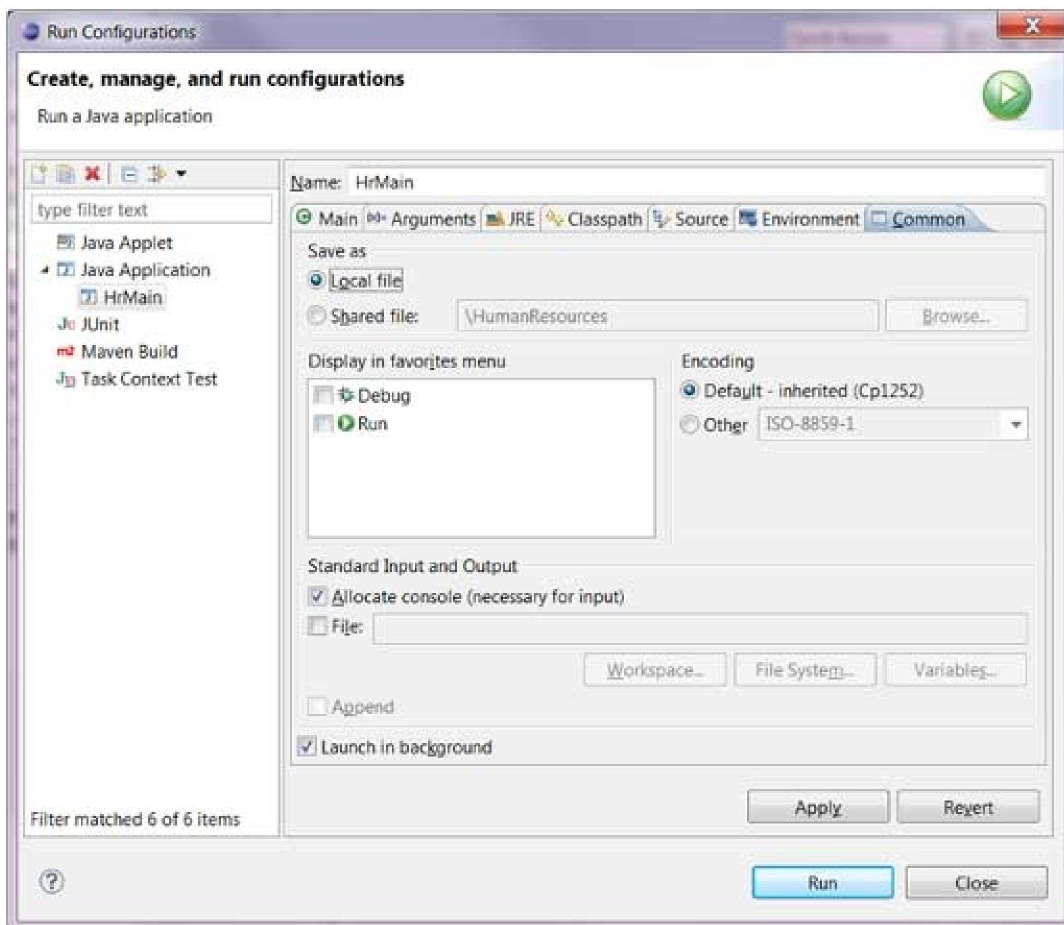
- 运行配置名称
- 项目名
- 主类名

Arguments(参数)项有：

- Program arguments(程序参数) 可以 0 个或多个
- VM arguments(Virtual Machine arguments:虚拟机参数) 可以 0 个或多个




Commons 选项卡中提供了通用配置，如标准输入输出的选项，可以到控制台或指定文件。




点击 Apply(提交) 按钮保存运行配置并点击 Run(运行) 按钮重新执行 Java 应用。

# Eclipse 运行程序

## 运行 Java 程序

我们可以在 Package Explorer 视图

可以在 Package Explorer 视图中快速运行 Java 程序。

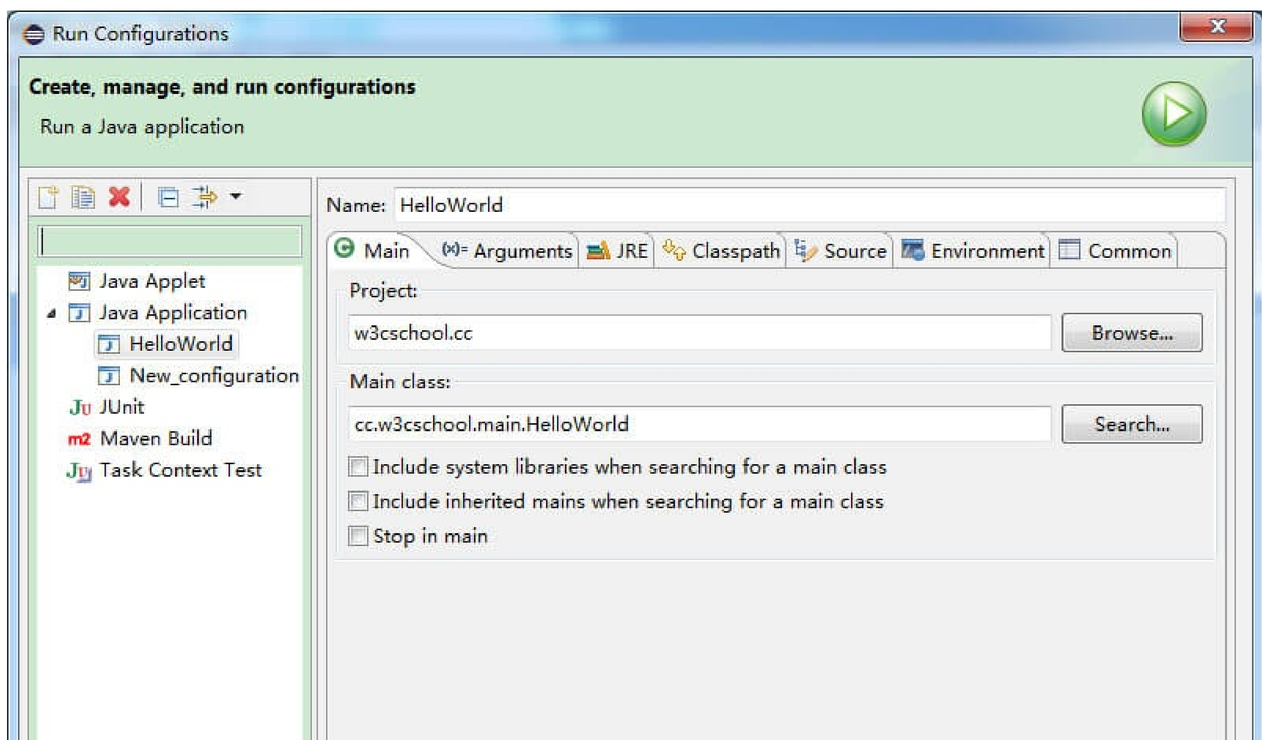
Package Explorer 视图:

鼠标右击包好 main 函数的 java 类选择 Run As > Java Application

同样你也可以在 Package Explorer 视图中选择包含 main 方法的类并按下快捷键：Alt + Shift + X, J

以下两种方式都能创建一个新的 Run Configuration(运行配置) 我们可以使用它来启动 Java 应用程序。

如果运行配置已经创建，你可以在 Run 菜单中选择 Run Configurations 来启动 Java应用，点击运行配置的名称，然后点击运行按钮的Java应用程序。




Run 菜单中的 Run 选项可以重新启动先前启动 Java 应用。




重新启动先前启动 Java 应用快捷键为 Ctrl + F11。

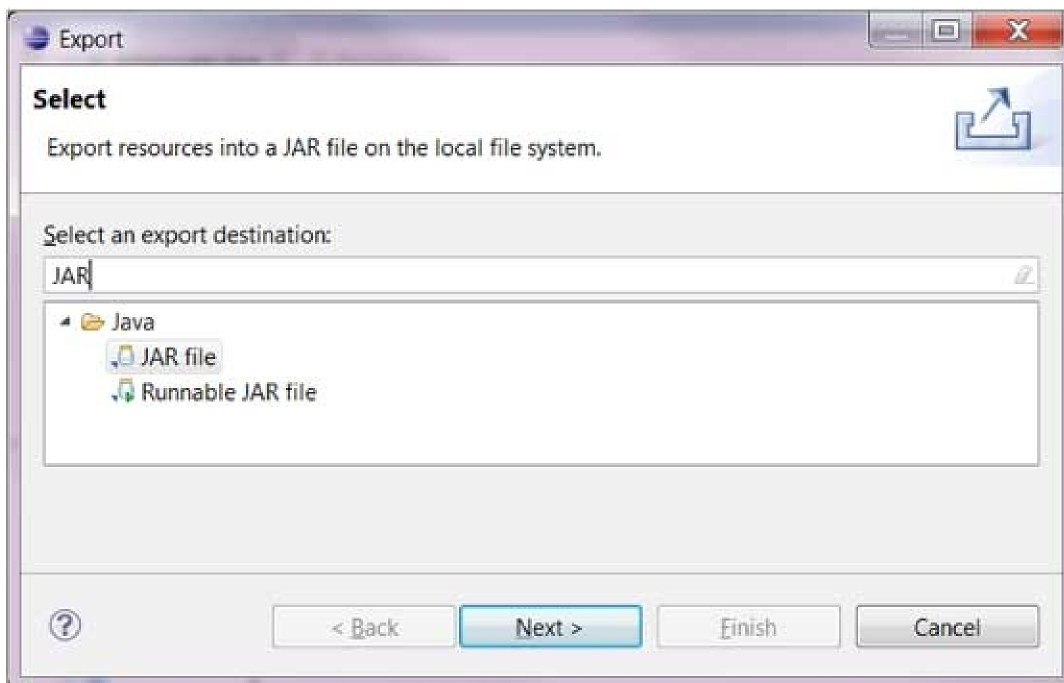
# Eclipse 生成jar包

## 打开 Jar 文件向导

Jar 文件向导可用于将项目导出为可运行的 jar 包。

打开向导的步骤为:

- 在 Package Explorer 中选择你要导出的项目内容。如果你要导出项目中所有的类和资源，只需选择整个项目即可。
- 点击 File 菜单并选择 Export。
- 在输入框中输入"JAR" 。

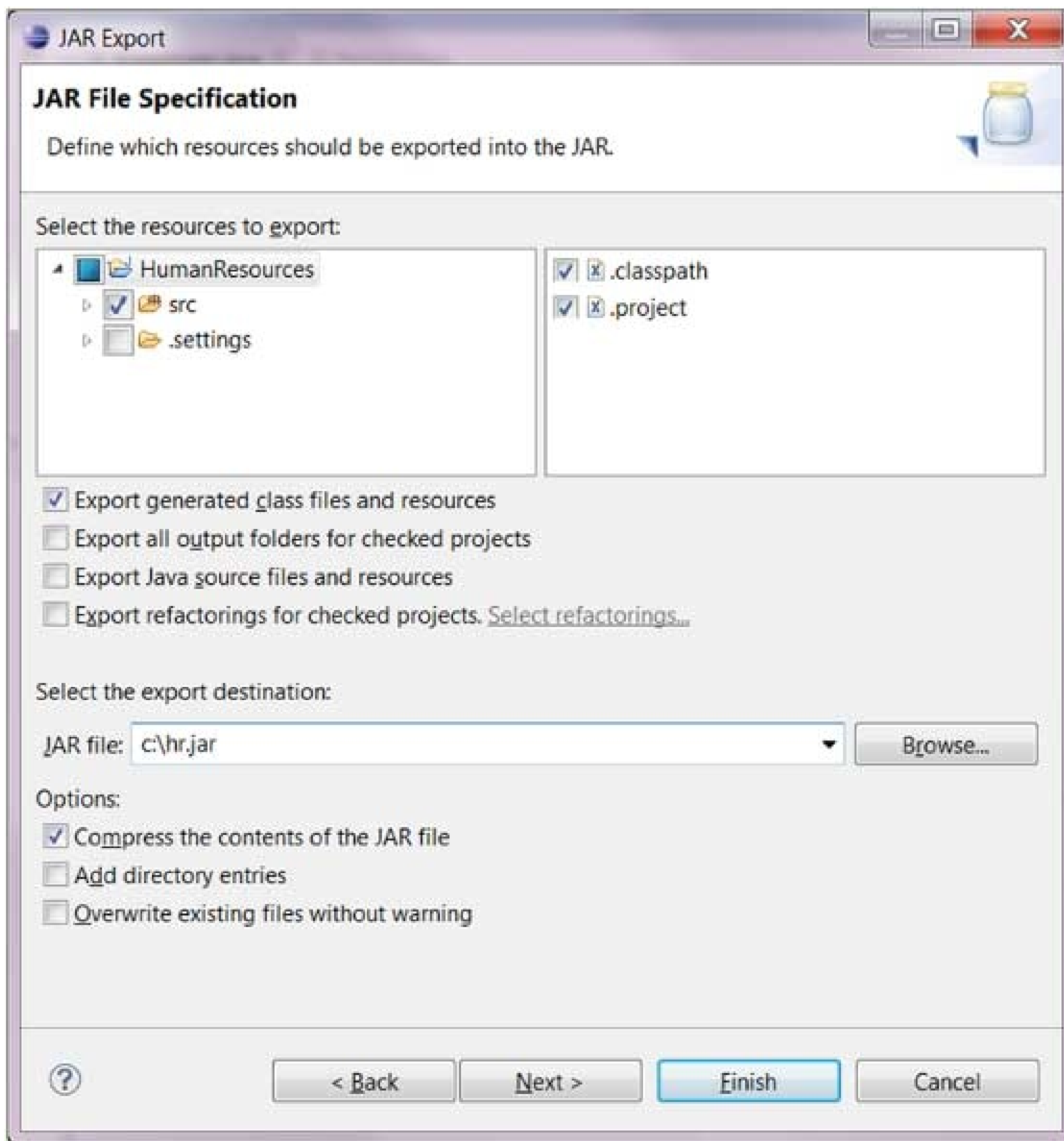



- 在选项中选择 JAR file 选项开启向导。
- 点击 Next 按钮

## 使用 Jar 文件向导

在 JAR File Specification（JAR 文件描述） 页面可进行以下操作：

- 输入 JAR 文件名及文件夹
- 默认只导出 class 文件。你也可以通过勾选 "Export Java source files and resources" 选项来导出源码的内容。




- 点击 Next 按钮修改 JAR 包选项
- 点击 Next 按钮修改 JAR Manifest 描述信息
- 点击 Finish 按钮完成操作

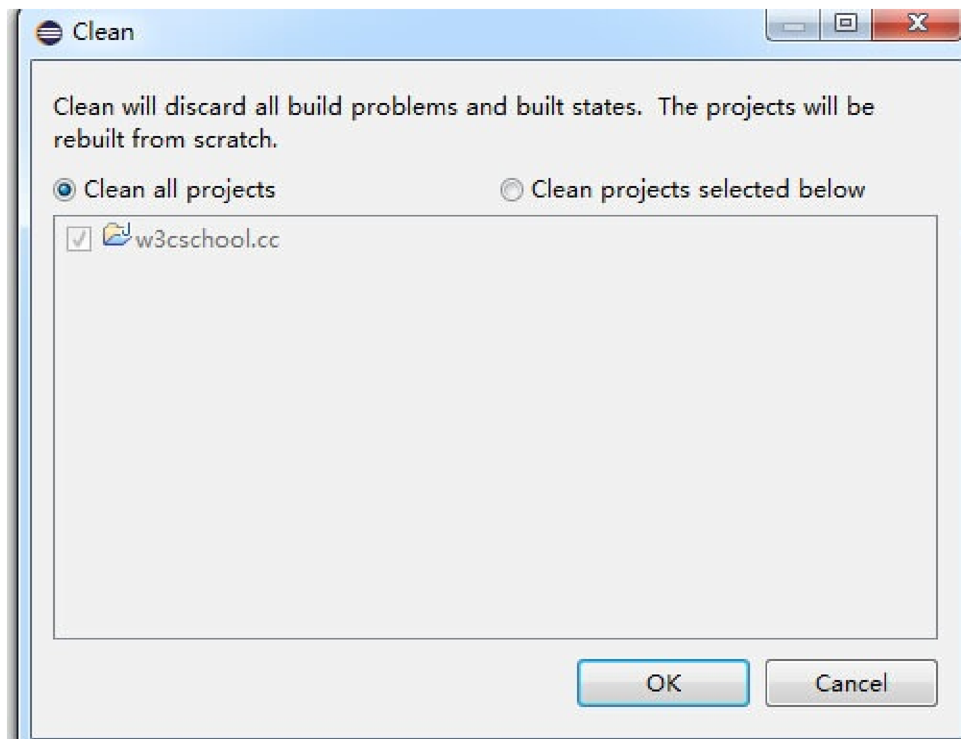
# Eclipse 关闭项目

## 为什么要关闭项目？

Eclipse 工作空间包含了多个项目。一个项目可以是关闭或开启状态。

项目打开过多影响有：

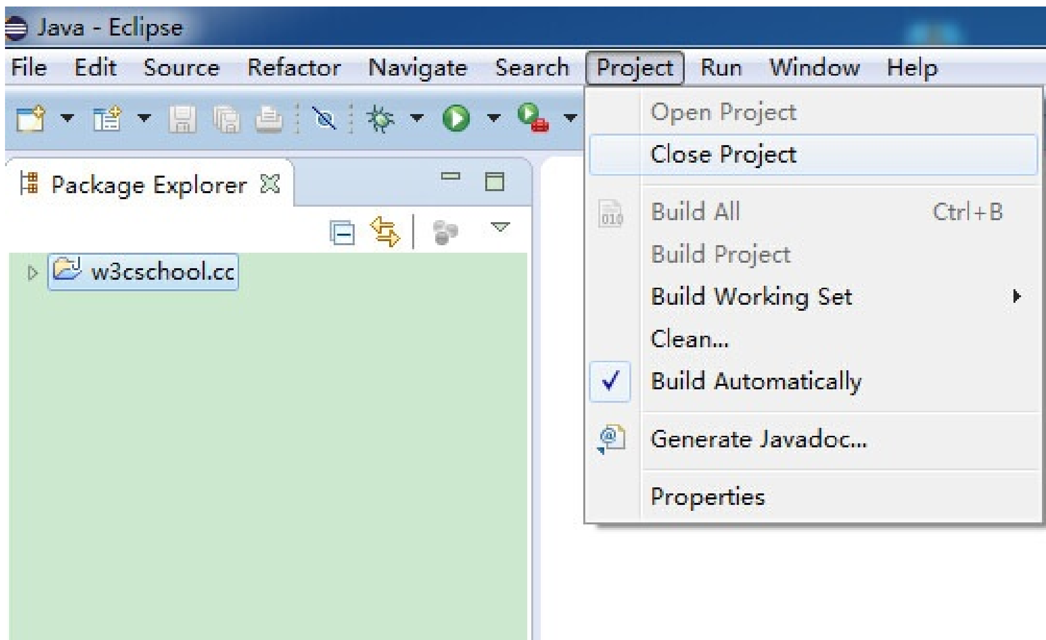
- 消耗内存
- 占用编译时间：在删除项目.class 文件（Clean All Projects）时并重新编译（在菜单上选择 Project > Clean > Clean all projects ）。




## 如何关闭项目?

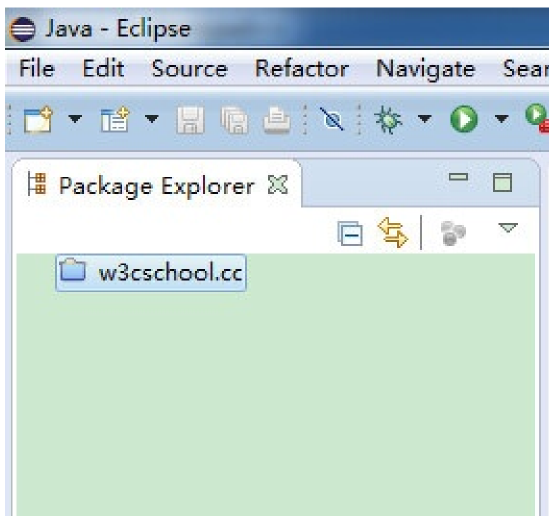
如果项目不处于开发阶段，我们就可以先关闭项目。

在 Package Explode 视图上选择要关闭的项目，并通过菜单上选择 Project > Close Project 来关闭项目。



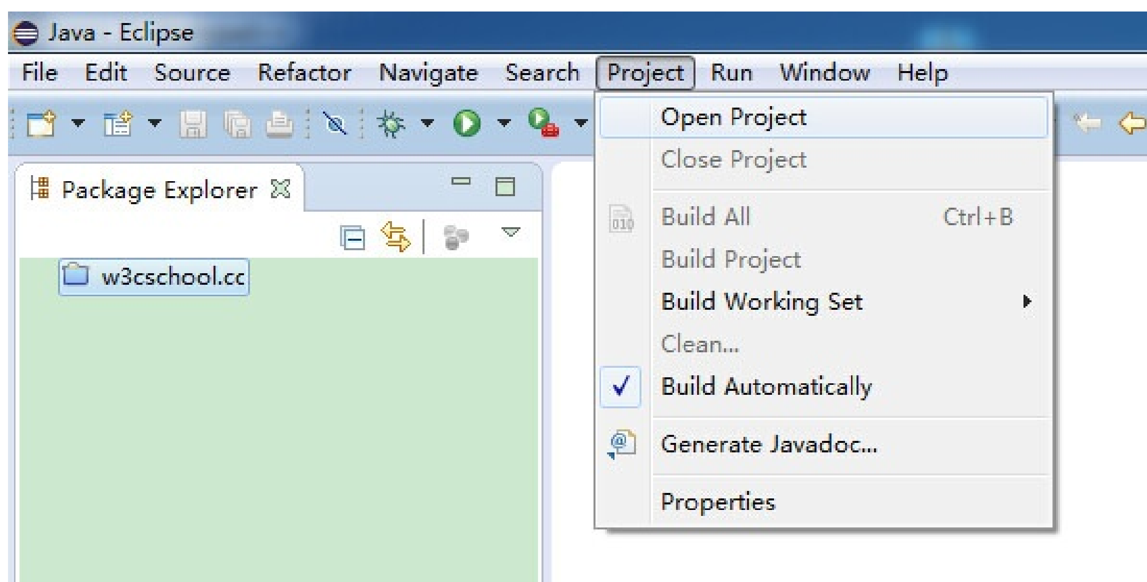

## 关闭后的项目

项目关闭后我们可以在 Package Explorer 视图看到项目的图标已经变了。关闭后的项目是不能编辑的。




## 重新开启项目

你可以通过选择 Project > Open Project。


Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Open Project
Close Project
Build All Ctrl+B
Build Project
Build Working Set
Clean...
Build Automatically
Generate Javadoc...
Properties
Package Explorer
w3cschool.cc

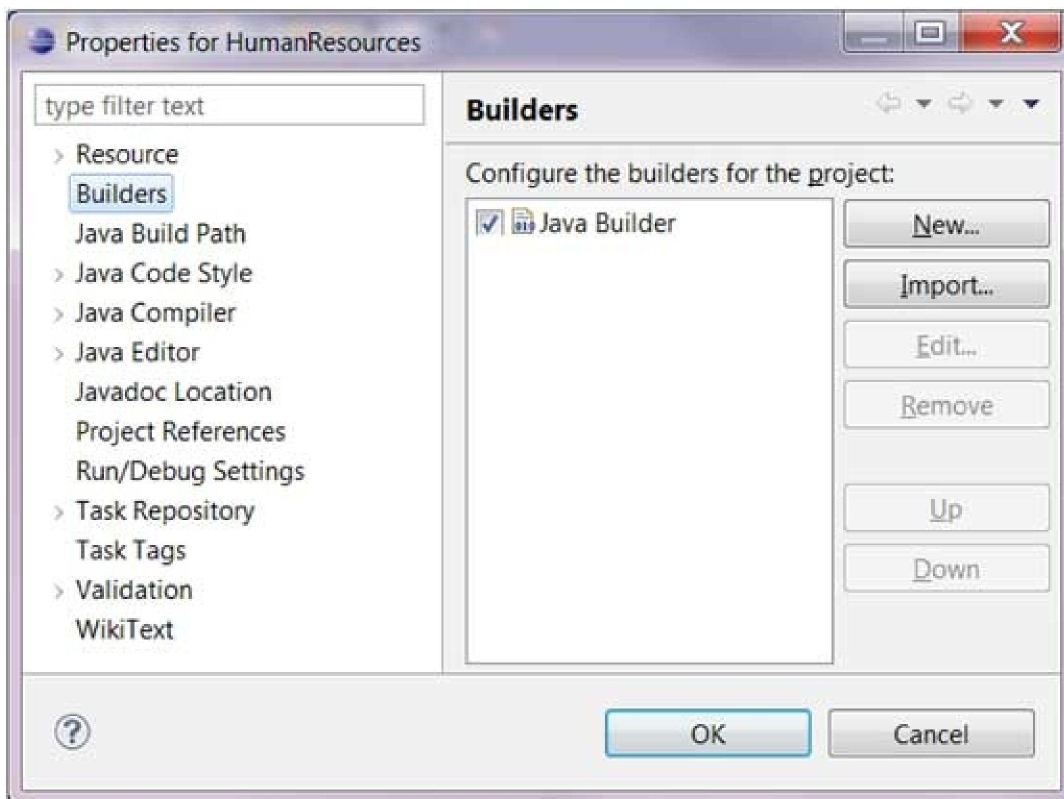
# Eclipse 编译项目

## 编译 Java 项目

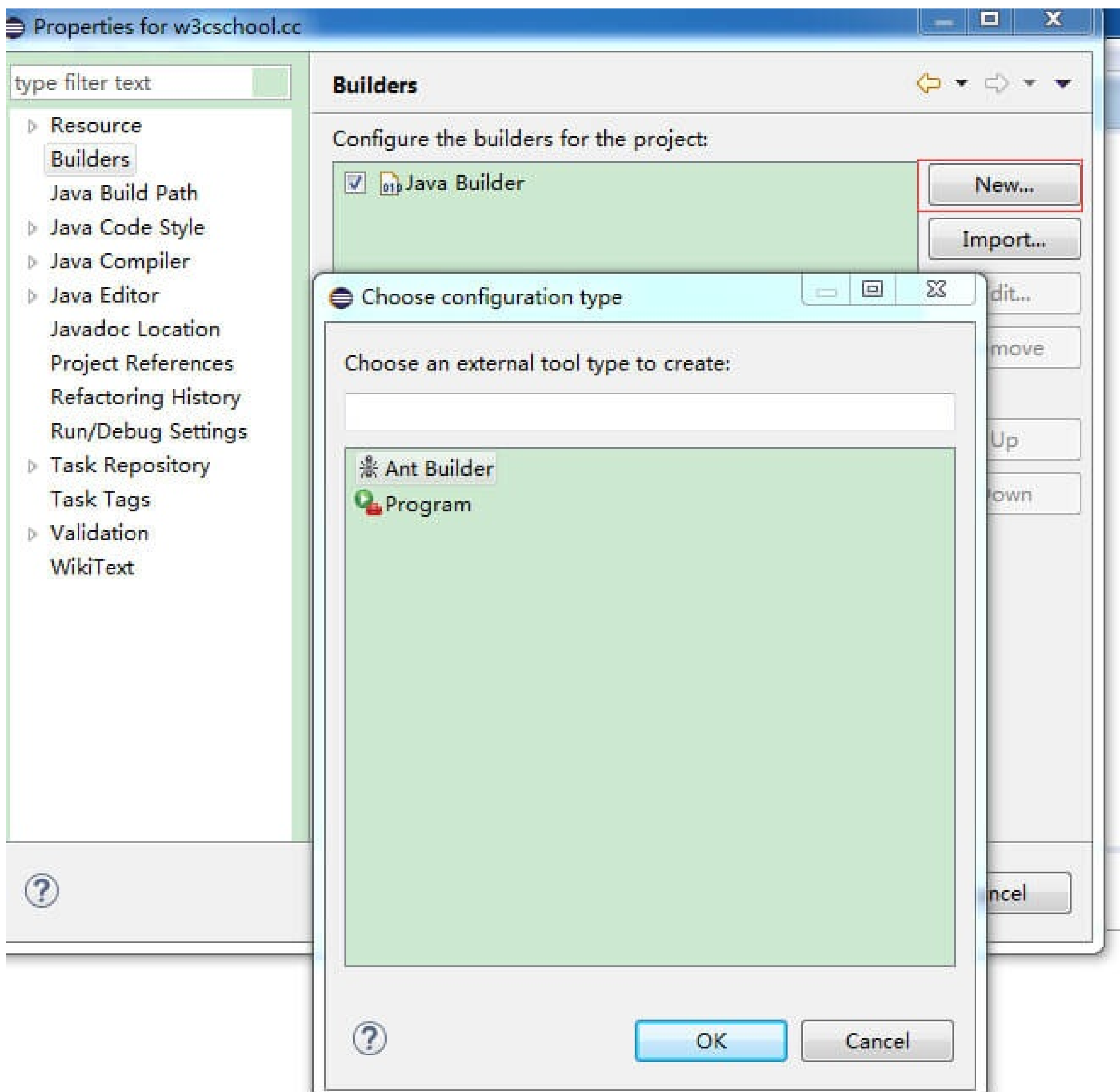
一个项目可以关联多个编译器。

java 项目关联的是 java 编译器。可以通过以下方式来查看项目关联的编译器：

- 在 Package Explorer 视图中鼠标右击项目并选择 Properties
- 在左侧的树形菜单中点击 Builders

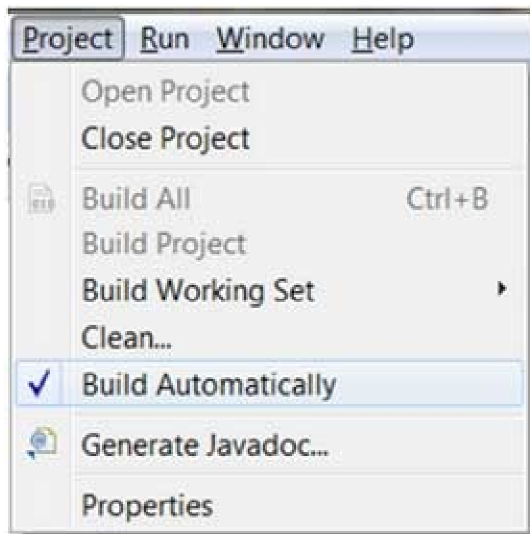



java编译器用于编译java项目。通过点 New 按钮我们可以让java项目关联 Ant builder 编译器。




java 编译器通过编译 java 项目生成 class 文件。当项目源码发生变化时会自动重新编译 java 代码。

可以通过去除 Project 菜单中 Build Automatically (自动编译)项来禁用自动编译功能。



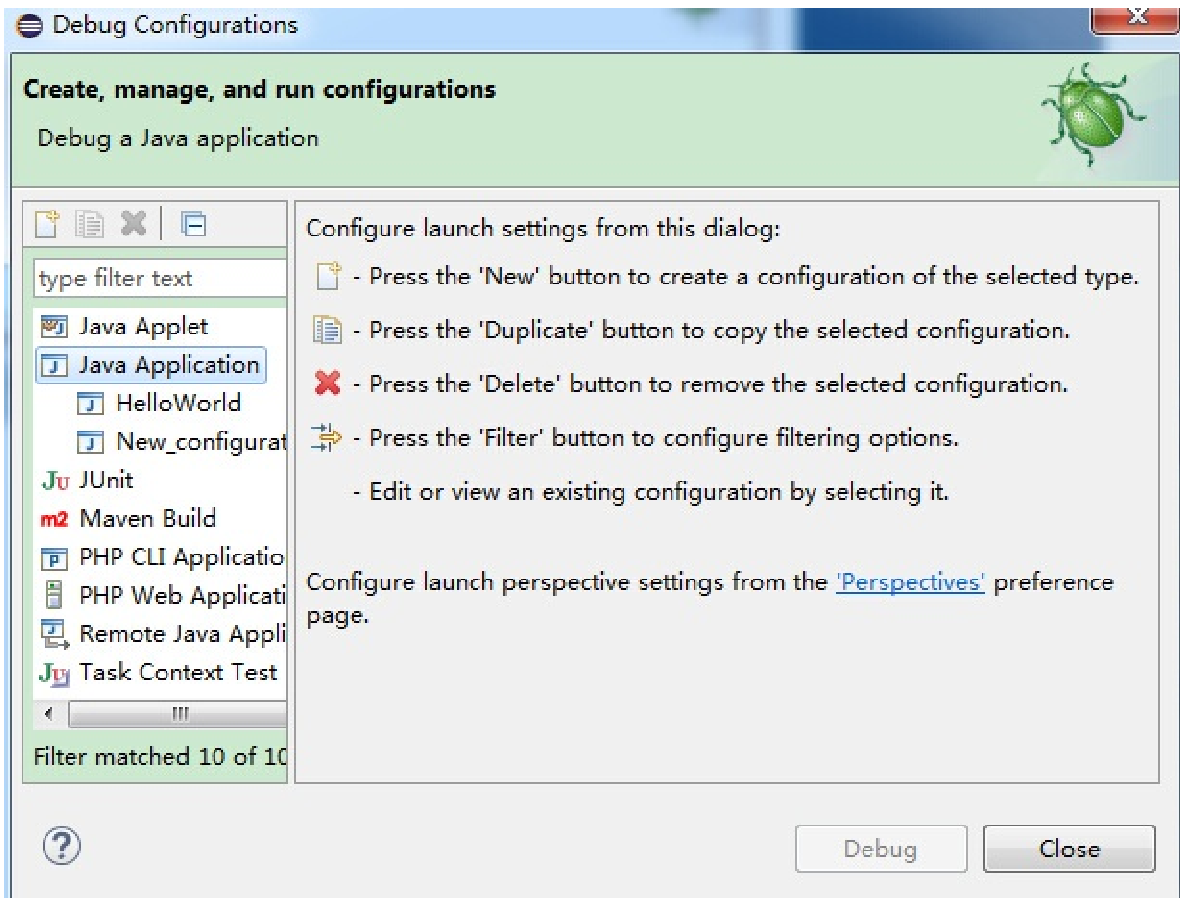

如果你禁用了自动编译功能，项目需要通过 Project 菜单中的 Build Project 菜单项来编译java项目。 如果勾选了 Build Automatically(自动编译) 项，则 Build Project(手动编译) 菜单项是不可用的。

# Eclipse Debug 配置

## 创建和使用 Debug 配置

Eclipse Debug 配置类似于运行配置但它是用于在调试模式下开启应用。

打开 Debug 配置对话框步骤为：Run > Debug Configurations 。



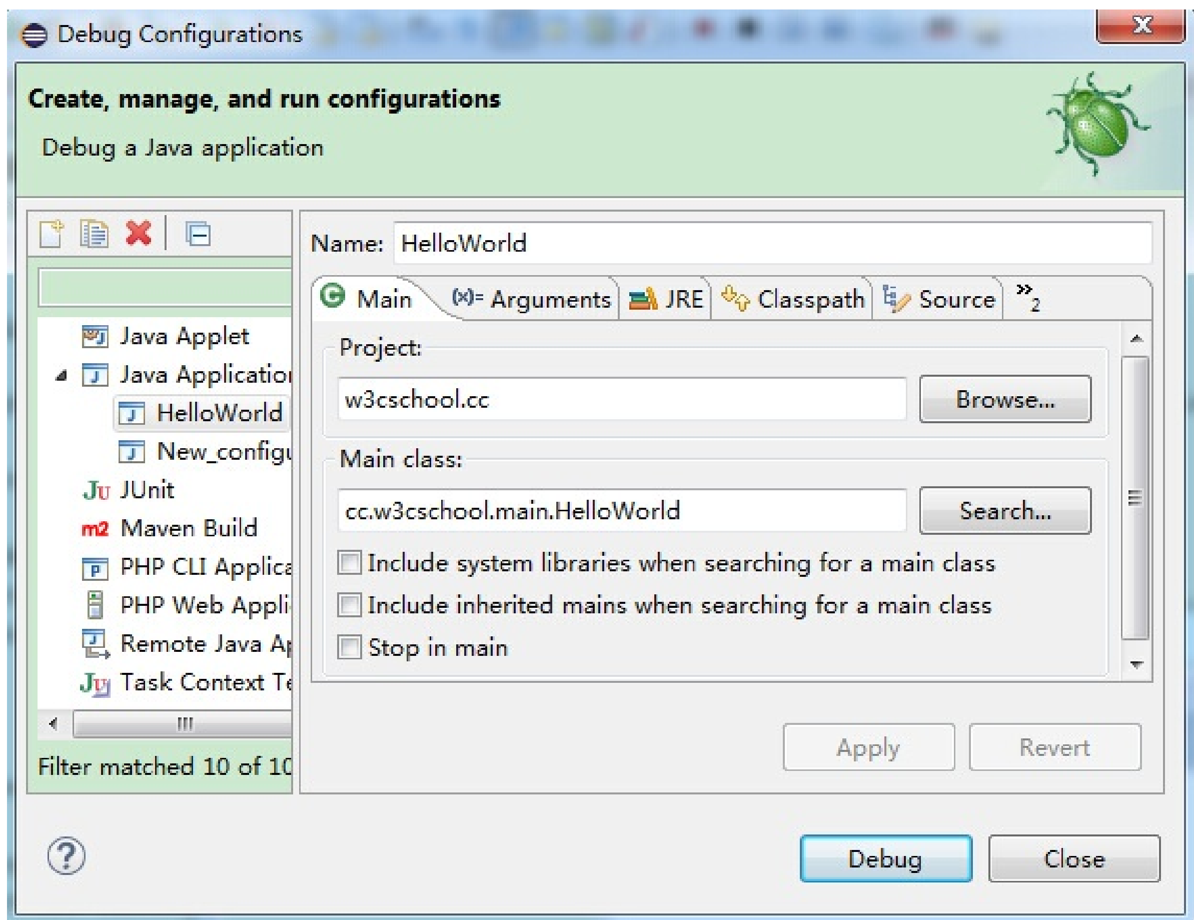

从左侧列表中选择 "Java Application" 选项来创建 Java 应用的调试配置并 New 按钮。

对话框中的描述信息有：

- 调试配置的名称
- 项目名称
- 主类名

arguments(参数)选项卡的描述信息有:

- 零个或多个程序参数
- 零个或多个虚拟机参数（VM arguments）




保存运行配置信息并点击 Apply 按钮，然后点击 Debug 按钮在调试模式下载入应用。

# Eclipse Debug 调试

## Debug 调试 Java 程序

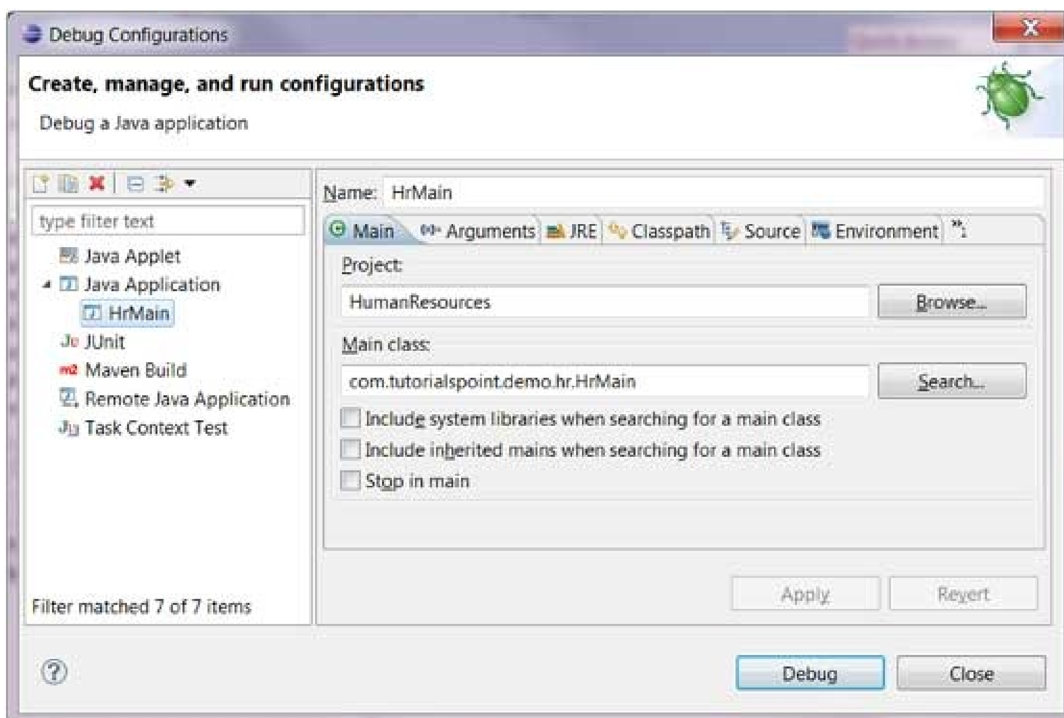
我们可以在 Package Explorer 视图调试 Java 程序，操作步骤如下：

- 鼠标右击包含 main 函数的 java 类
- 选择 Debug As > Java Application

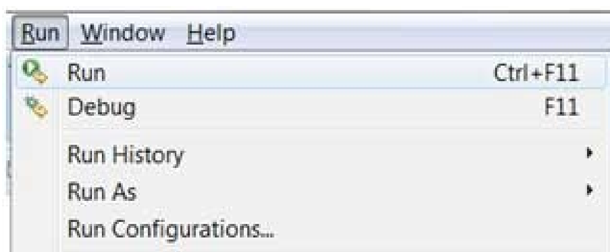
该操作也可以通过快捷键来完成，快捷键组合为 Alt + Shift + D, J。

以上操作会创建一个新的 Debug Configuration（调试配置），并使用该配置来启动 Java 应用。

如果 Debug Configuration（调试配置）已经创建，你可以通过 Run 菜单选择 Debug Configurations 选取对应的类并点击 Debug 按钮来启动 Java 应用。



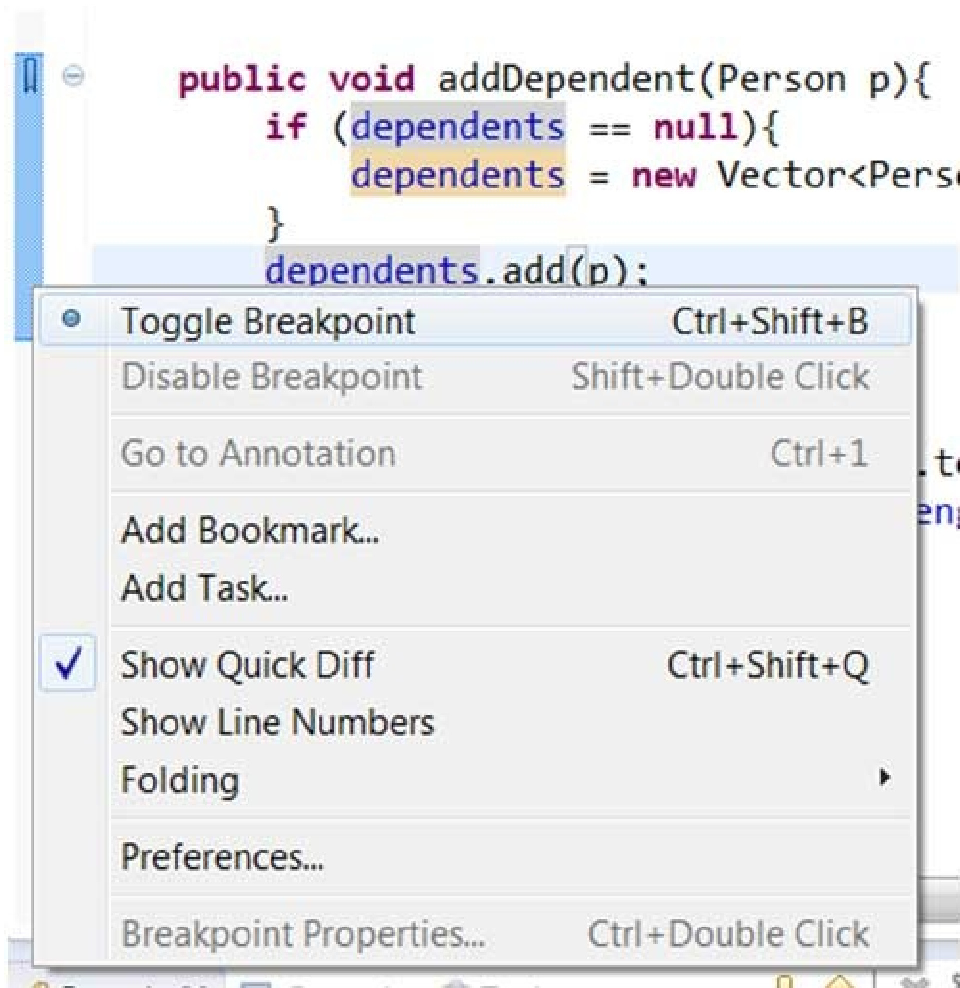

Run 菜单的 Debug 菜单项可以重新加载之前使用了调试模式的 java 应用。




重新加载之前使用了调试模式的 java 应用快捷键为 F11。

当使用调试模式开启java程序时，会提示用户切换到调试的透视图。调试透视图提供了其他的视图用于排查应用程序的故障。

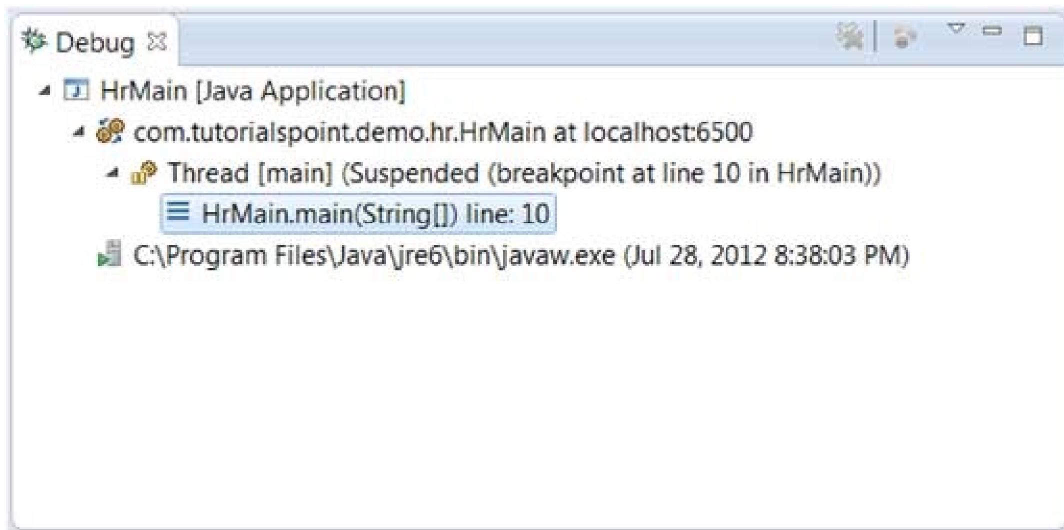
java 编辑器可以设置断点调试。 在编辑器中右击标记栏并选择 Toggle Breakpoint 来设置断点调试。



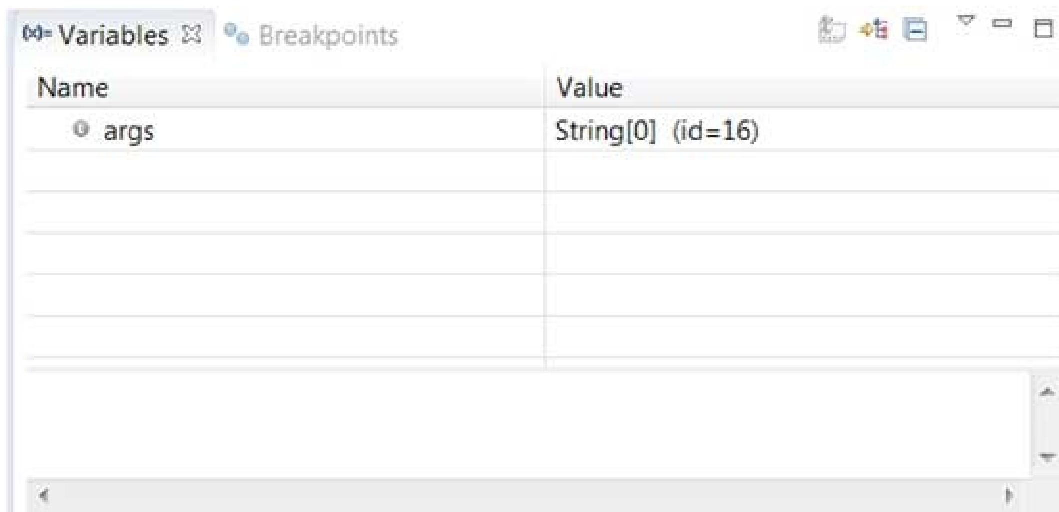

断点可以在标记栏中看到。也可以在 Breakpoints View（断点视图）中看到。

当程序执行到断点标记的代码时 JVM 会挂起程序，这时你可以查看内存使用情况及控制程序执行。

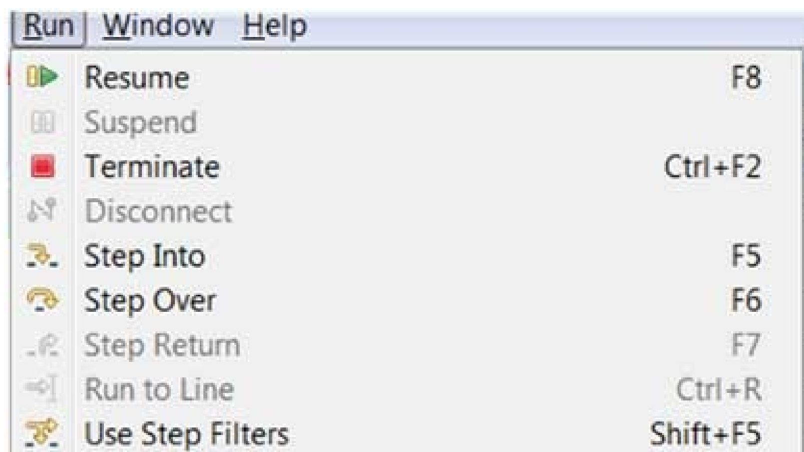
程序挂起时，Debug(调试)视图可以检查调用堆栈。




variables(变量)视图可以查看变量的值。




Run 菜单中有继续执行(Resume)菜单项，跳过(Step Over)一行代码，进入函数(Step Into)等。



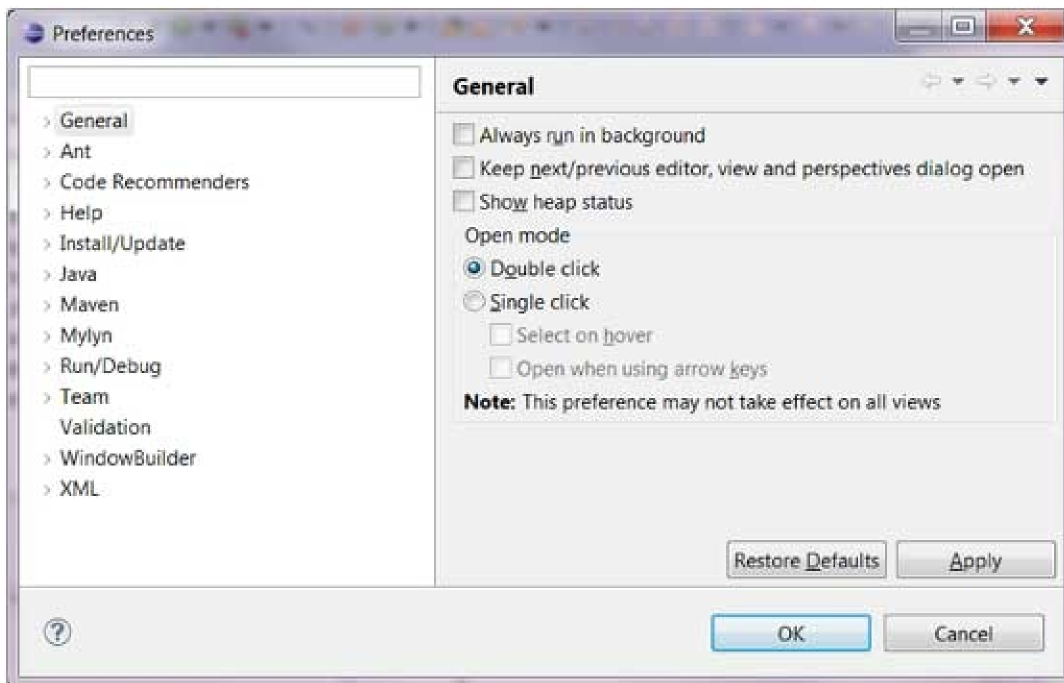

以上图片中显示了 Resume, Step Into 和 Step Over 等关联的快捷键操作。

# Eclipse 首选项(Preferences)

## 设置首选项

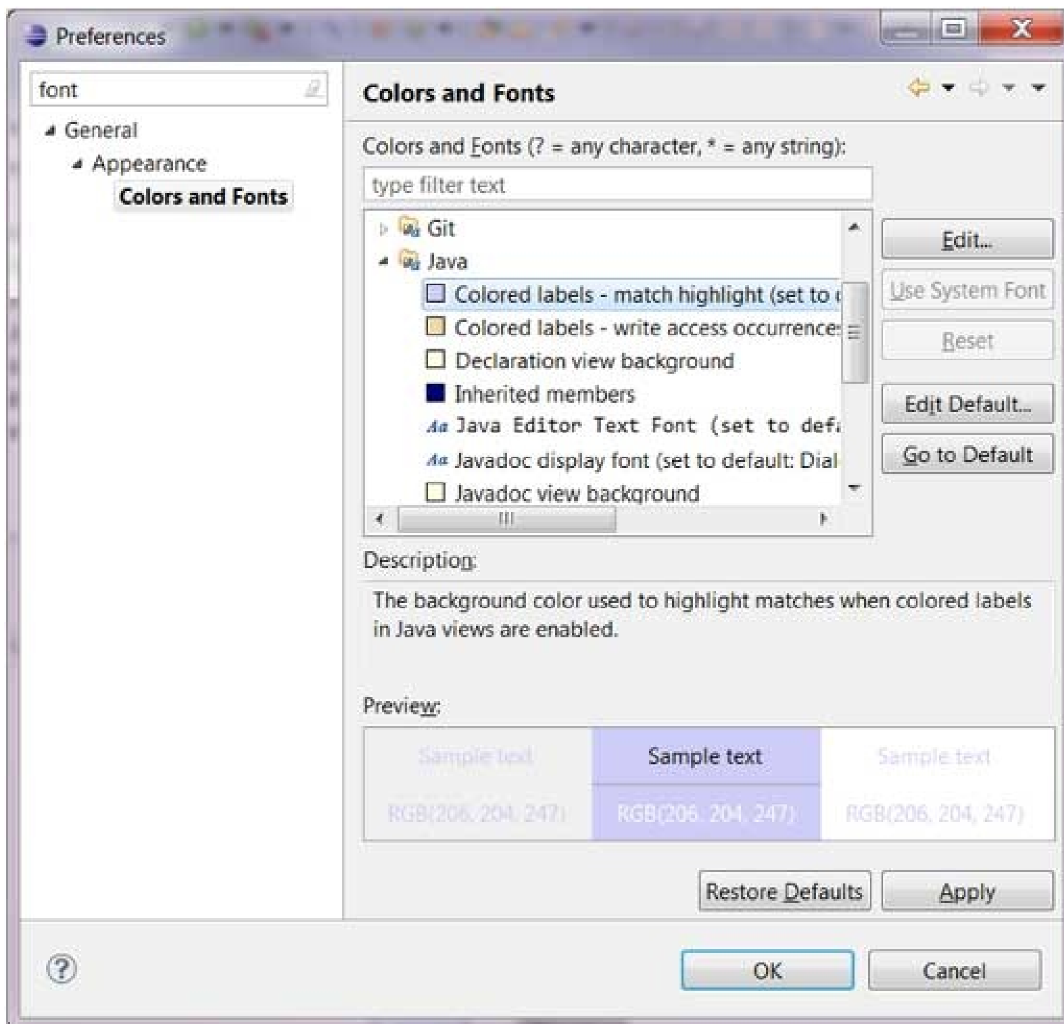
该对话框可通过框架管理但是其他插件可以设置其他页面来管理首选项的配置。

我们可以通过 Window 菜单选择 Preferences 菜单项来开启该对话框。




首选项页面有多个分类组成。你可以在左侧菜单中展开各个节点来查看首选项的配置。

左上角的输入框可以快速查找首选项页面。 你只需在输入框中输入要查找的首选项页面的字母即可快速找到对应的首选项页面。 例如：输入 font 即可查找到 Font(字体) 首选项页面。



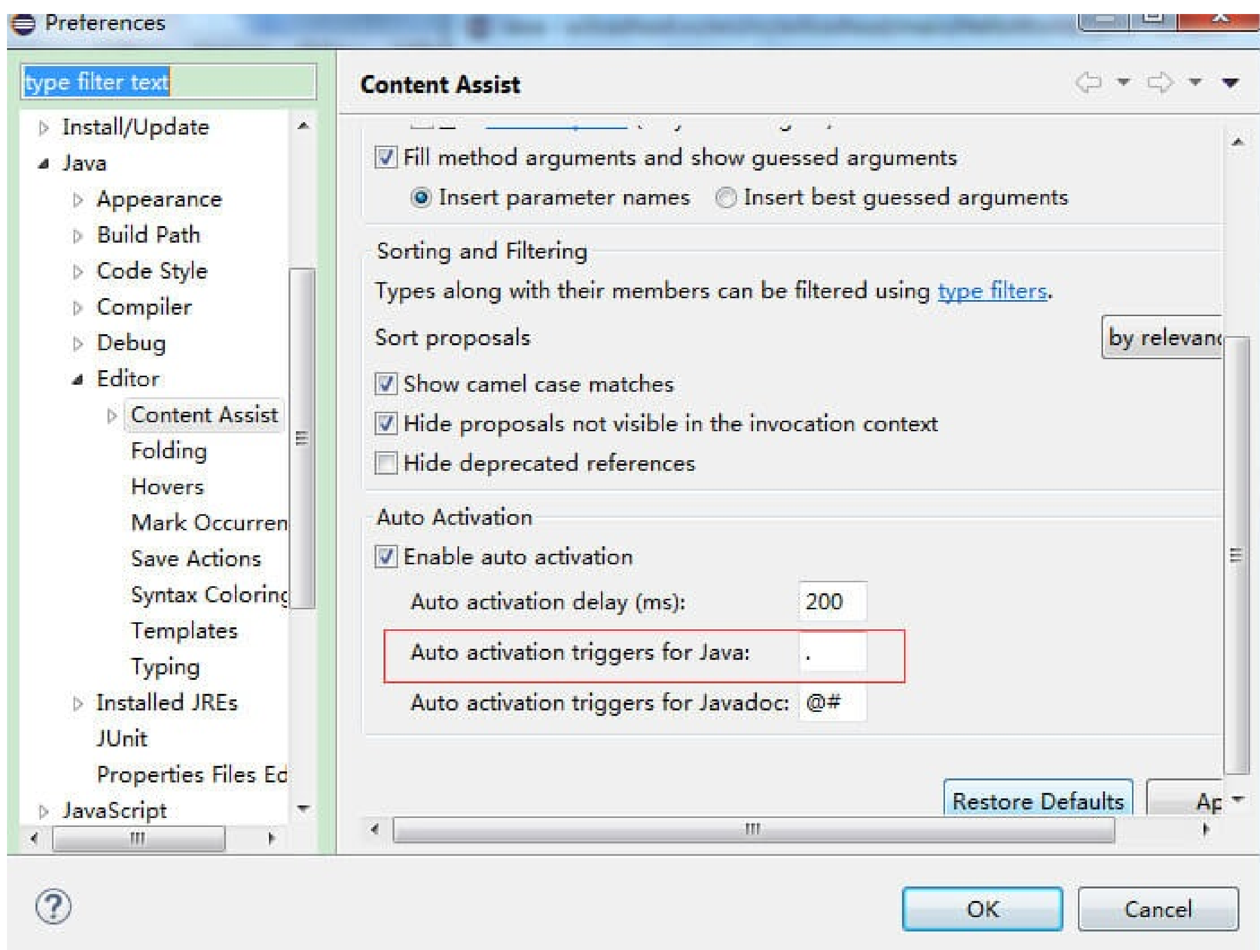

在你完成首选项页面的配置后点击 OK 按钮就可以保存配置，点击 Cancel 按钮用于放弃修改。

# Eclipse 内容辅助

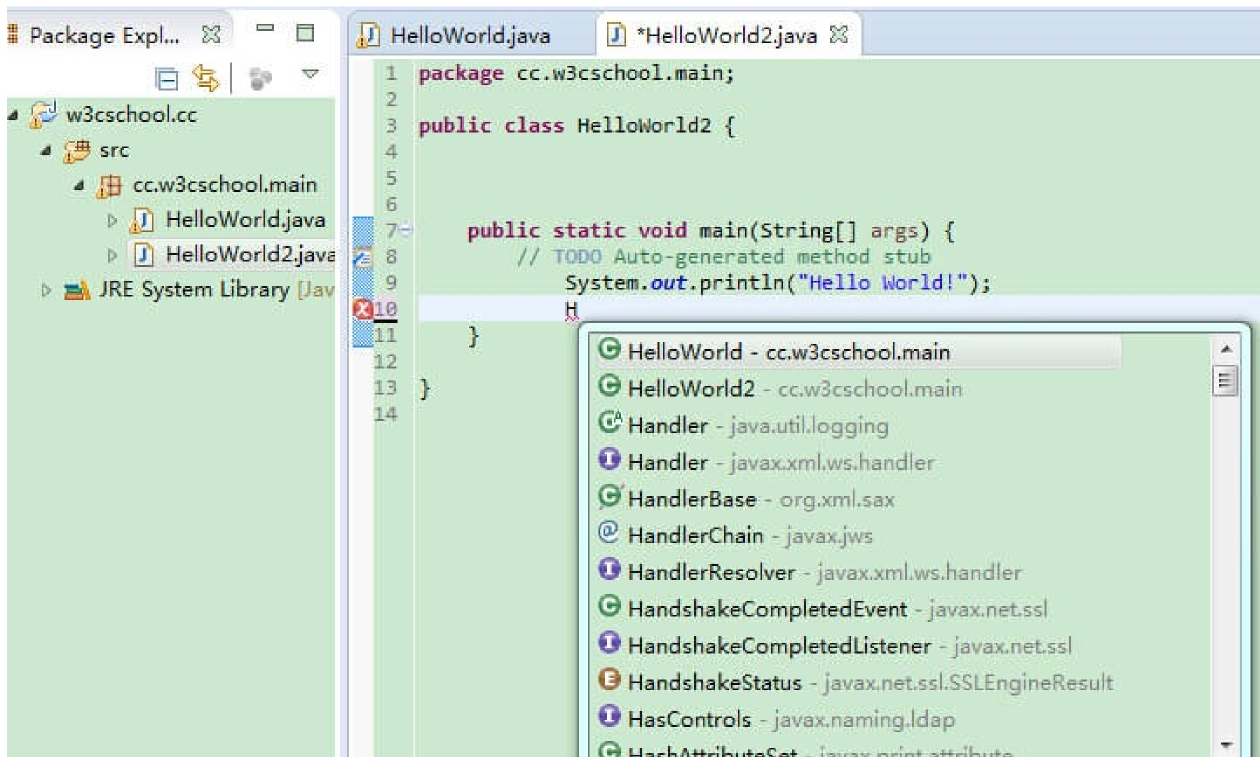
## 使用内容辅助

Eclipse中我们可以使用代码提示来加快开发速度，默认是输入**"."**后出现自动提示，用于类成员的自动提示。

设置自动提示的配置在：window->Preferences->Java->Editor->Content Assist：

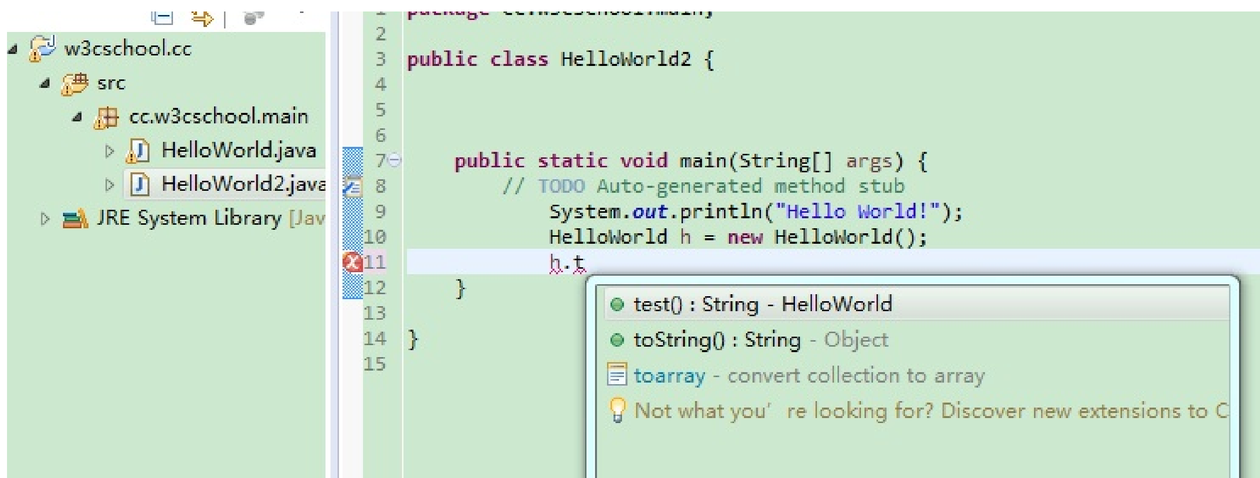



如果能在我们输入类的首字母按 **alt + /** 后就出现自动提示，。




输入 "." 后出现自动提示的内容有：

- 类变量
- 类方法
- 超类方法
- 其他相关类



# Eclipse 快速修复

## 使用快速修复

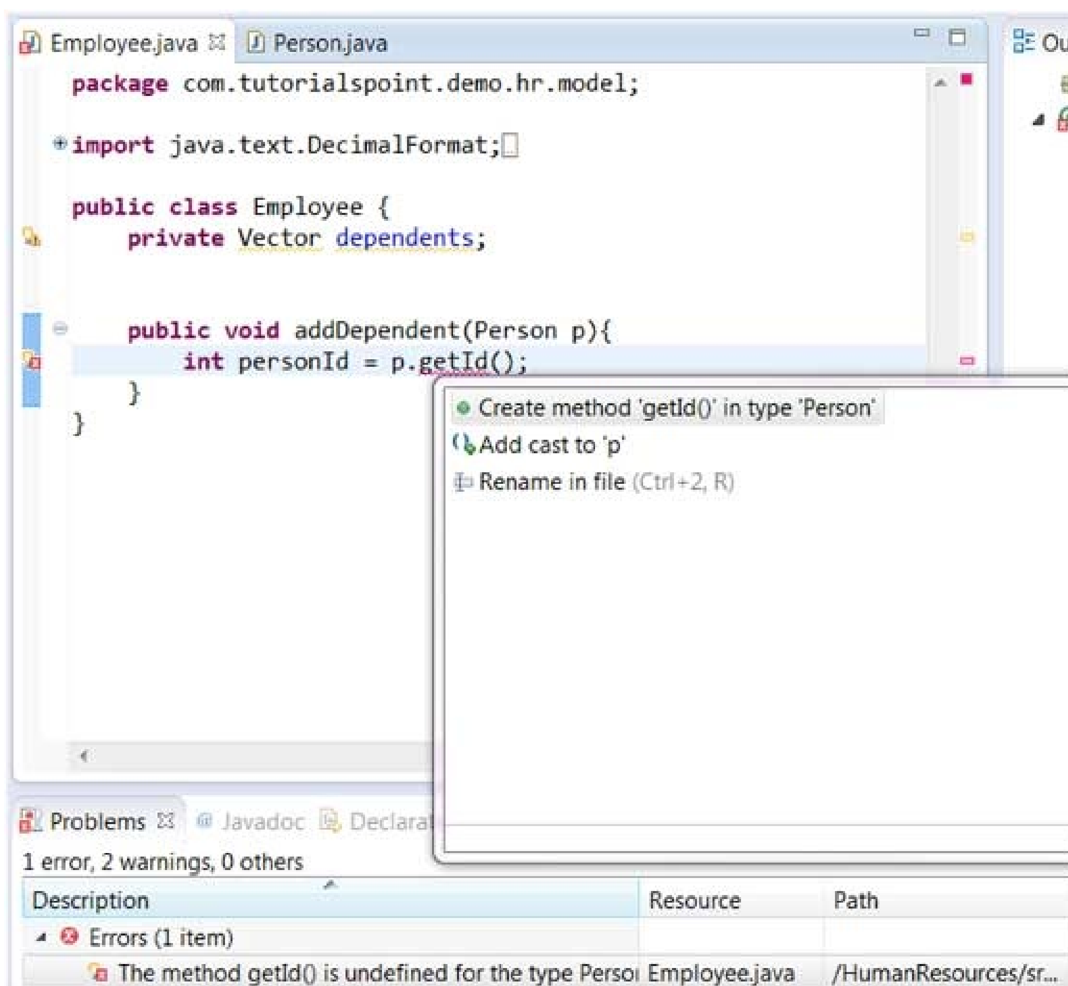
在 Eclipse 编辑器中当你输入字母时，编辑器会对你输入的内容进行错误分析。

Java 编辑器中使用 Java 语法来检测代码中的错误。当它发现错误或警告时：

- 使用红色波浪线突出错误
- 使用黄色的波浪线突出警告
- 在 Problem 视图中显示错误和警告
- 在垂直标尺上显示黄色小灯泡及警告和错误标识

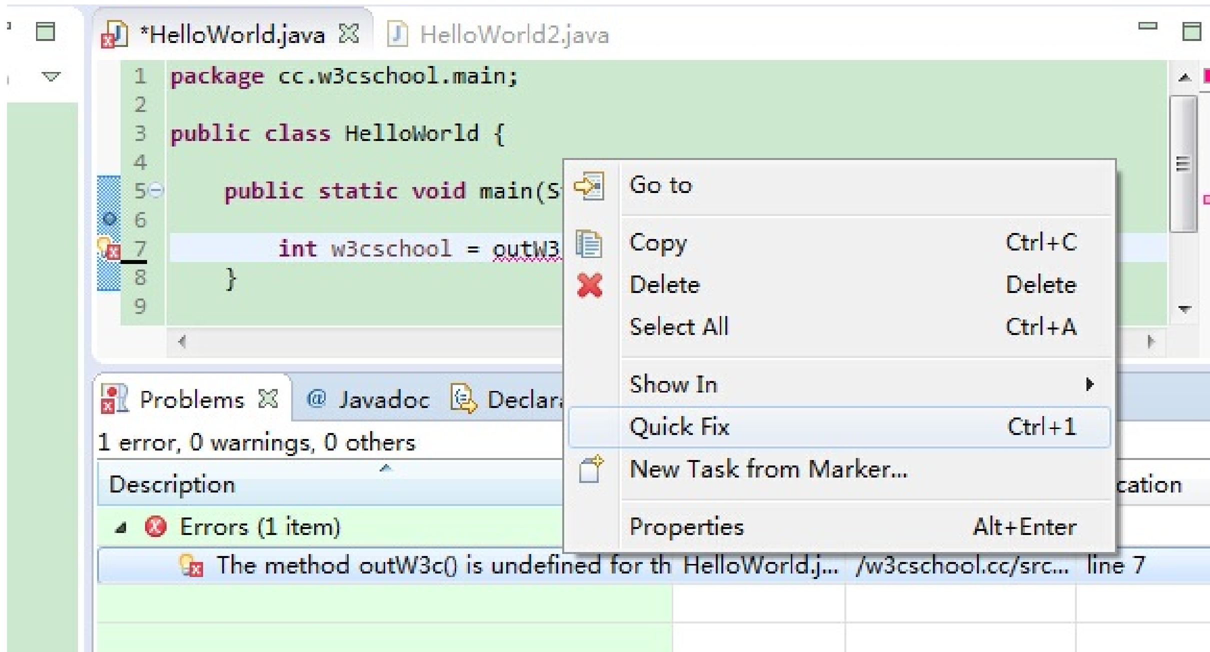
快速修复的对话框提供了解决的方案。快速修复对话框可通过以下方式调用：

- 将鼠标指针放在波浪线上
- 点击小灯泡
- 将鼠标指针放在突出的文本上并选择 Edit 菜单上的 Quick fix 项或者按下快捷键 Ctrl + 1



在上图中，getId 被高亮显示，因为 Person 类中没有一个名为的 getId() 方法。 在弹出的修复方案中选择 "Create method 'getId()' in type 'Person'" 这样就能在 Person 类中添加 getId() 方法。

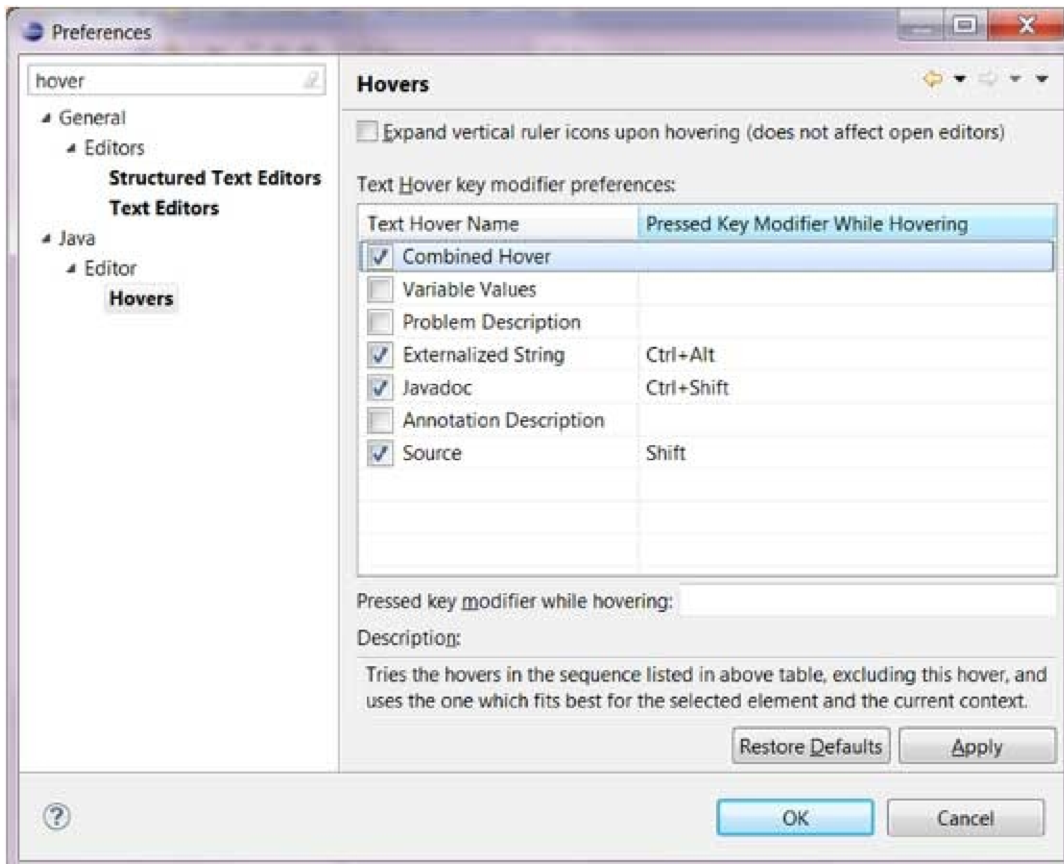
也可以通过右键点击 Problems 视图中的错误项，然后选择快速修复菜单项显示的快速修复对话框，如下图所示：



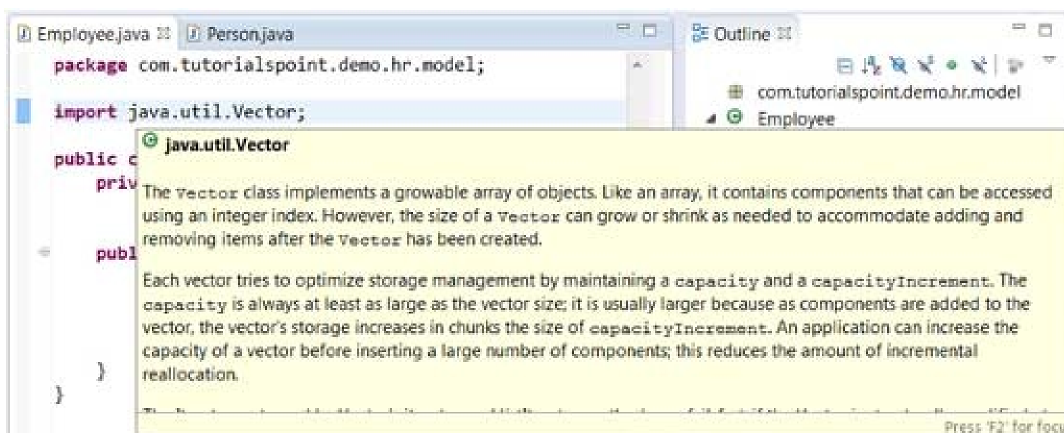
# Eclipse 悬浮提示

## 使用悬浮提示

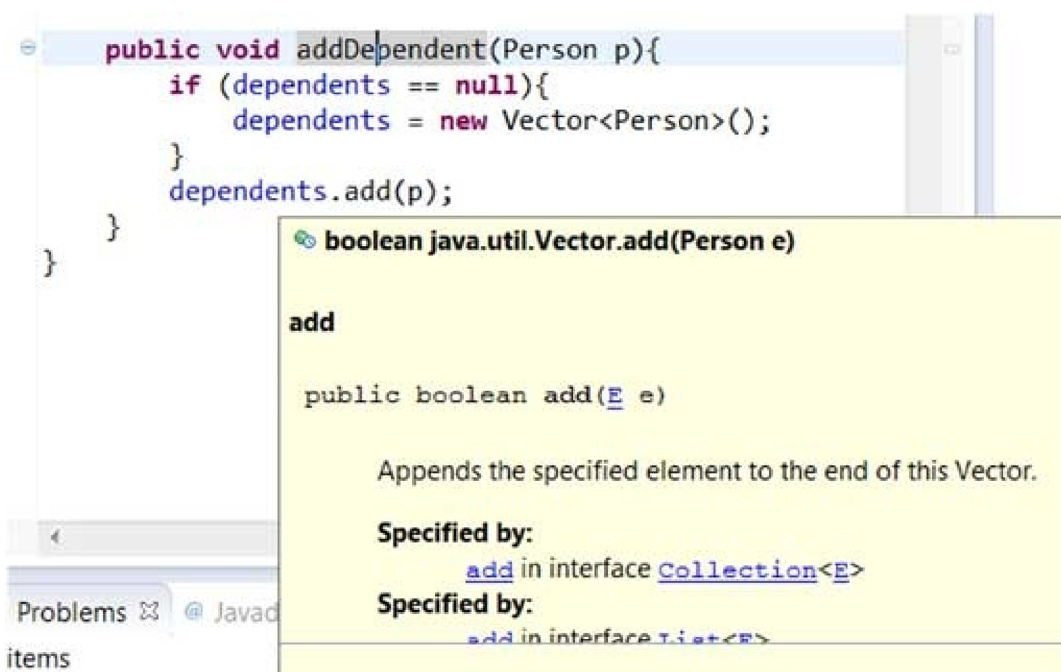
java 编辑器中包含了不同类型的悬浮提示，悬浮提示提供了鼠标指针指向元素的额外信息。所有java编辑器中相关的悬浮提示可以通过 preference(首选项) 的 Hovers 页面来配置（搜索框中输入 "hover"）。




java 编辑器中将鼠标指针移至类上，将显示与该类相关的java文档信息。




java 编辑器中将鼠标指针移至方法上，将显示与该方法相关的java文档信息。


public void addDependent(Person p){
if (dependents == null){
dependents = new Vector<Person>();
}
dependents.add(p);
}
}
boolean java.util.Vector.add(Person e)
add
public boolean add(E e)
Appends the specified element to the end of this Vector.
Specified by:
add in interface Collection<E>
Specified by:
Problems
items

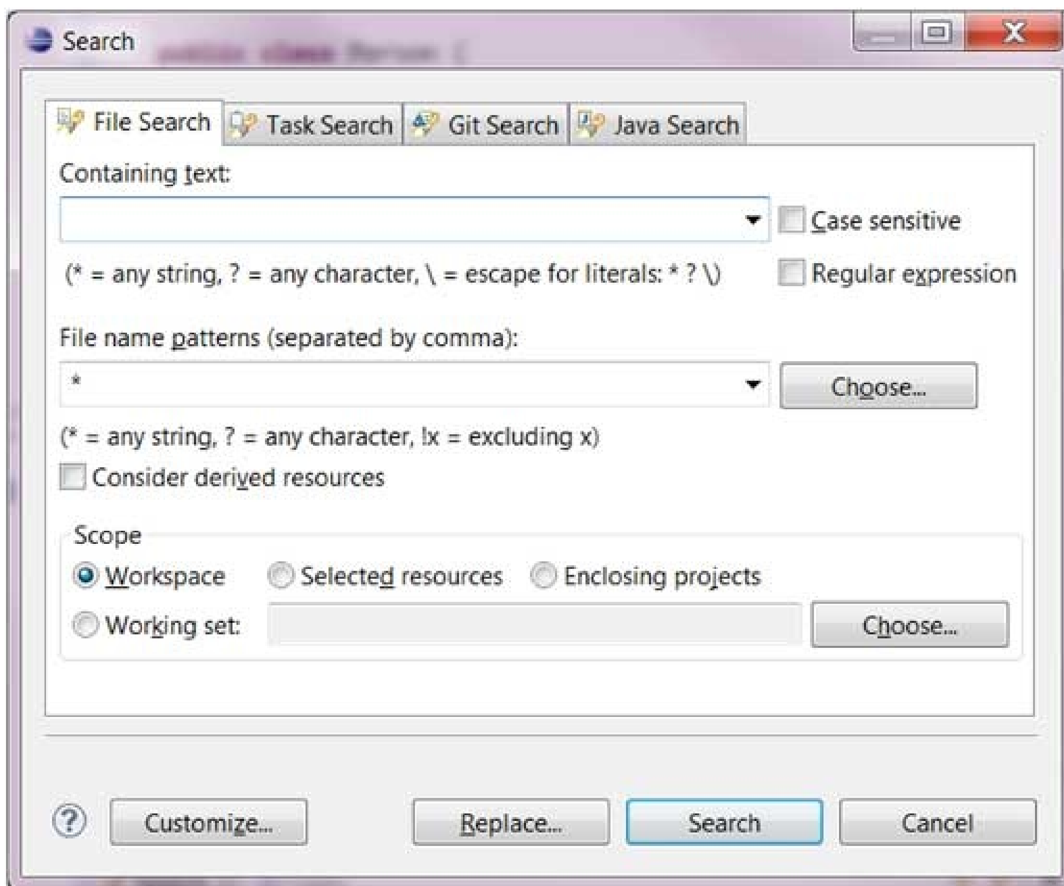
# Eclipse 查找

## 工作空间中查找

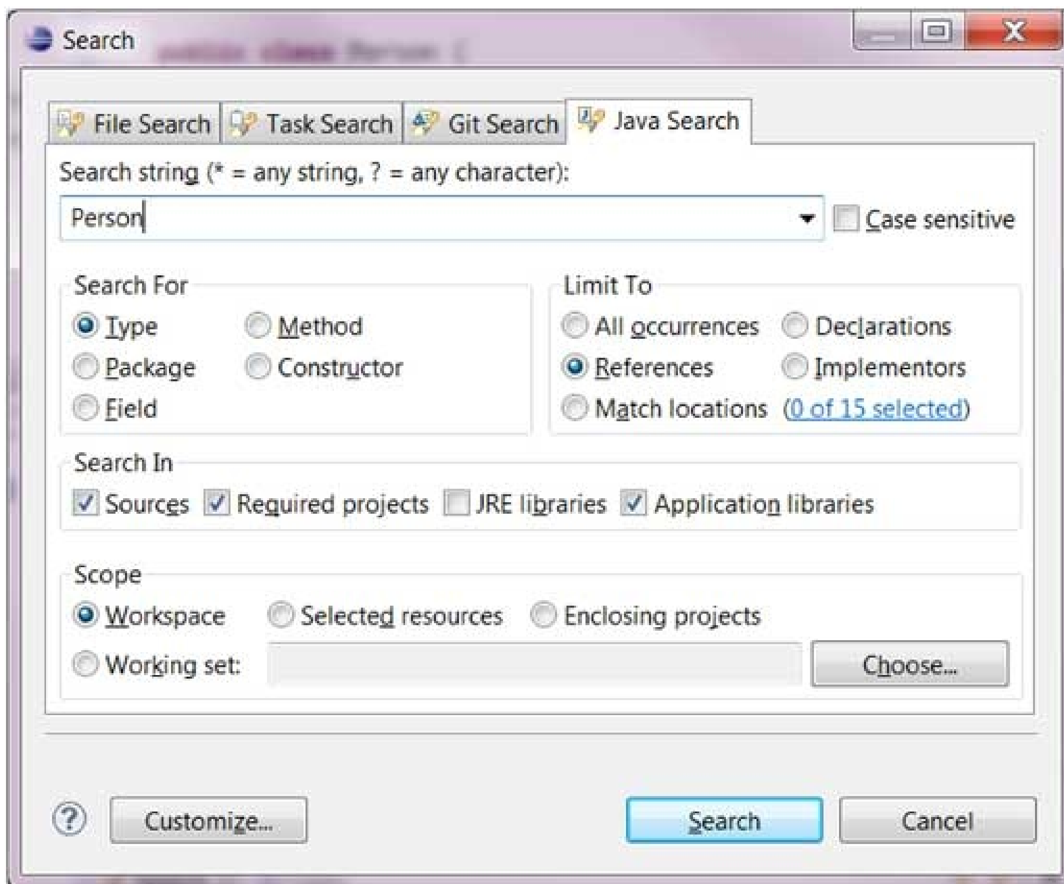
Eclipse 查找对话框中可以允许用户在指定工作空间上使用单词或字母模式来查找文件。 或者你可以在指定项目或在 package explorer 视图上选择好指定文件夹来查找。

可通过以下方式来调用查找框：

- 在 Search 菜单上选择 Search 或 File 或 Java
- 按下快捷键： Ctrl + H



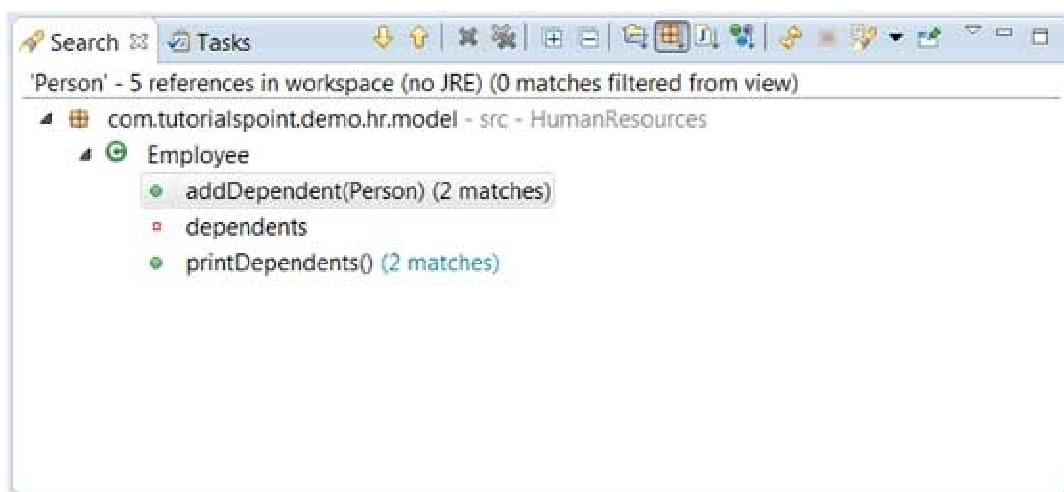

文件(File)查找允许用户查找所有文件类型，而 Java 查找只针对 Java 文件进行查找。




例如我们查找 Person 类型使用的情况，可以通过 Java 查找页面：

- 在查找框中输入 Person
- 在 search for 的单选按钮中选择 Type
- 在 limit to（限于）单选按钮中选择 References
- 点击 Search

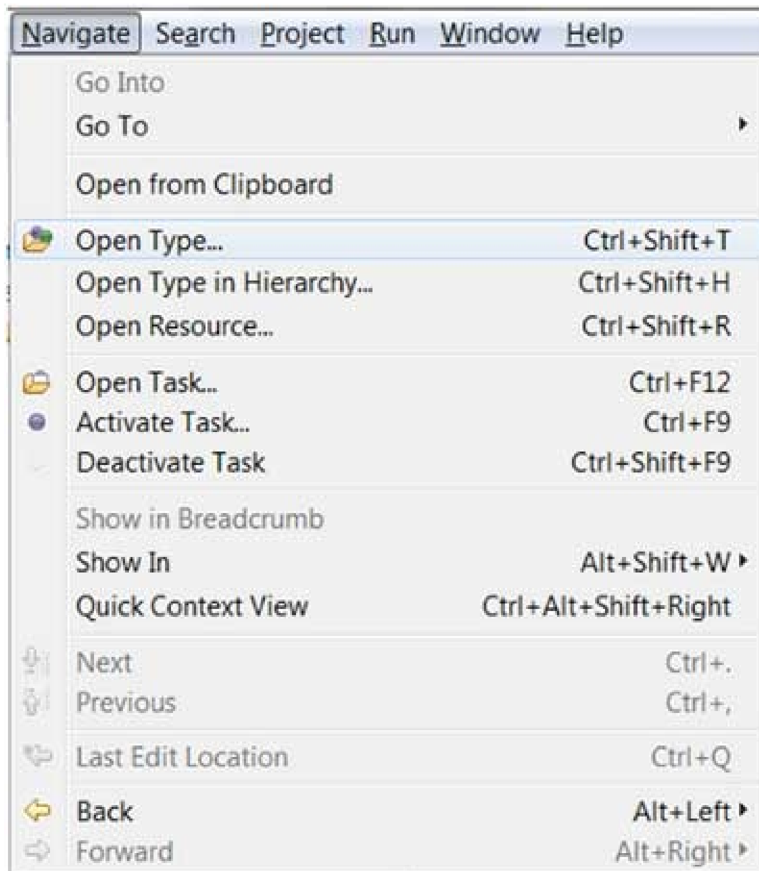
Search 视图中显示结果如下：



# Eclipse 浏览(Navigate)菜单

## 浏览 Eclipse 工作空间

浏览(Navigate)菜单提供了多个菜单可以让你快速定位到指定资源。

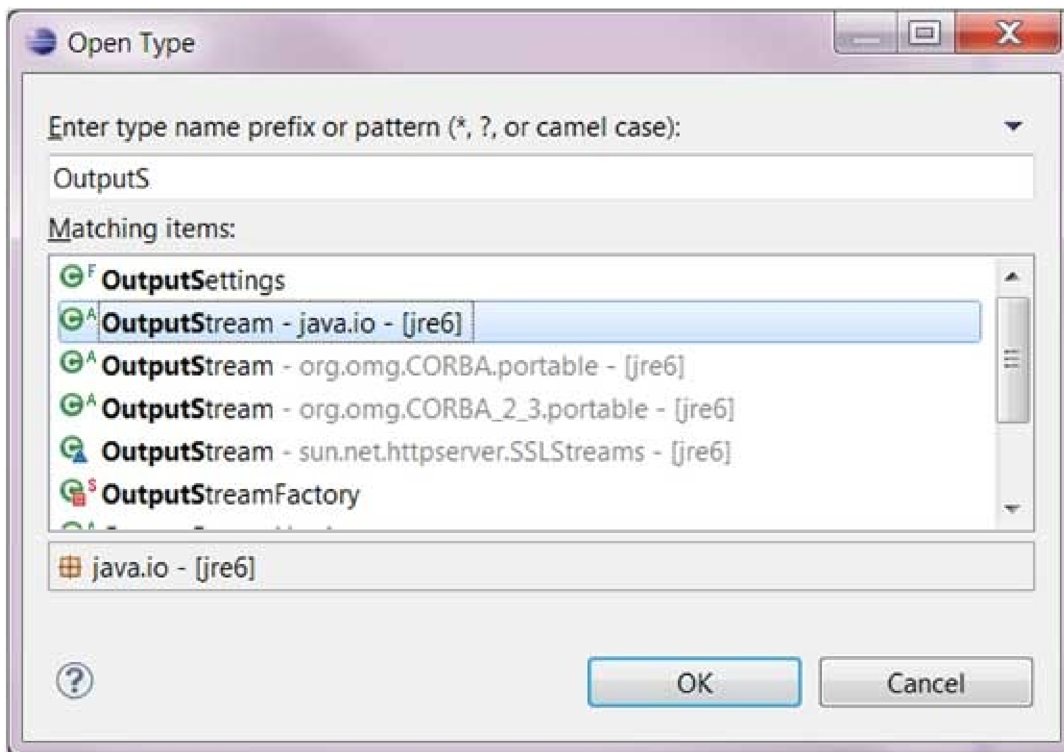



上图中 Open Type, Open Type in Hierarchy 和 Open Resource 三个菜单项是非常有用的。

## Open Type

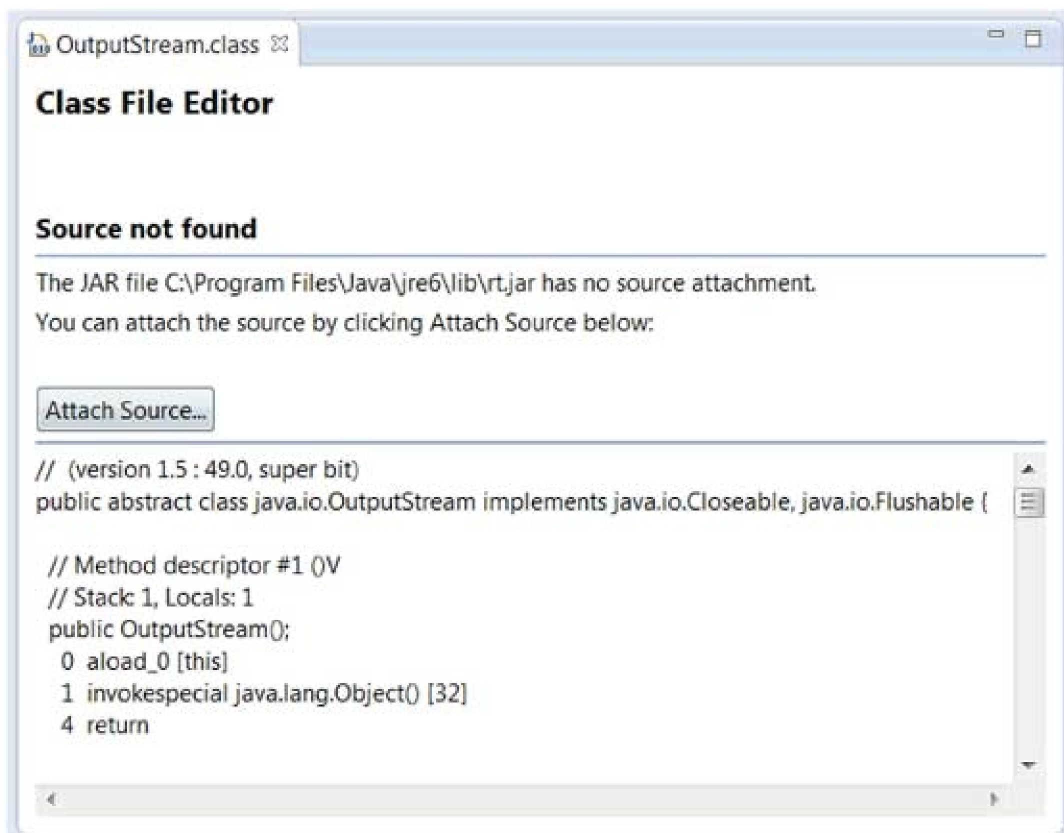
Open Type 菜单项可以打开一个对话框，对话框中可以查找 Java 类型文件。

你可以在输入框中输入类名查找。 '*' 号表示 0 个或多个字母，'?' 号表示单个字母可用于指定模式。对话框中将显示所有匹配的模式。




你列表中选择你查找的文件即可。

Eclipse 将打开一个编辑器，显示所选择的类型。如果所选类型不能显示源代码，将使用类文件编辑器显示所选类型的字节码。

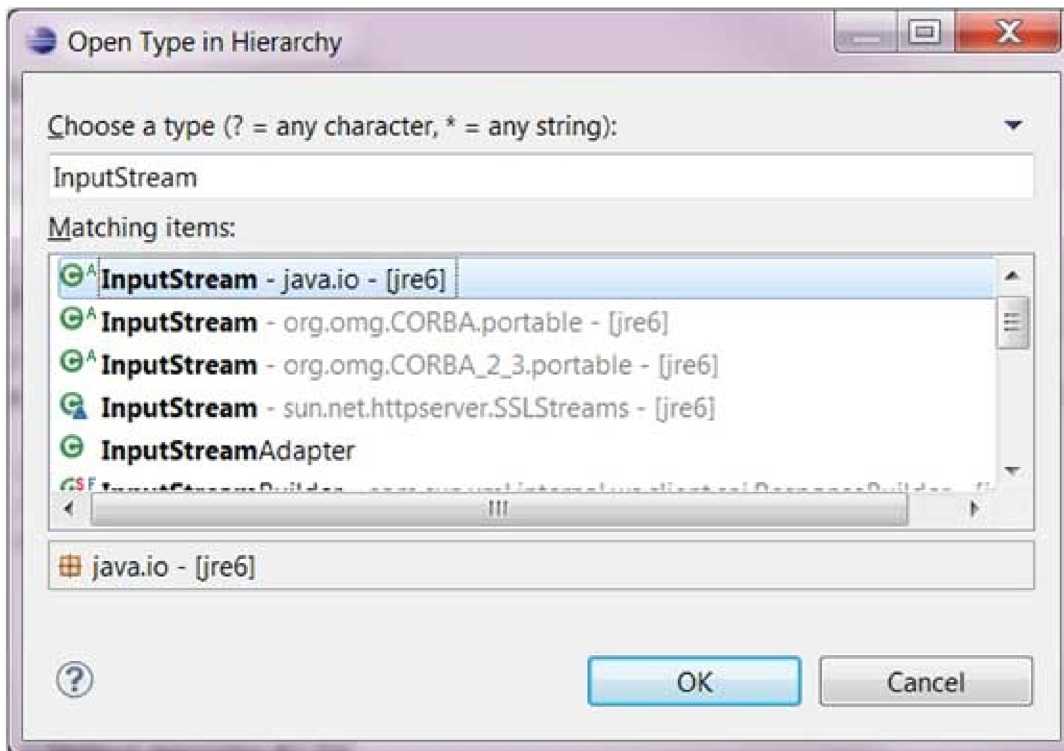



你可以点击 Attach Source 按钮来查看类文件对应的源码。

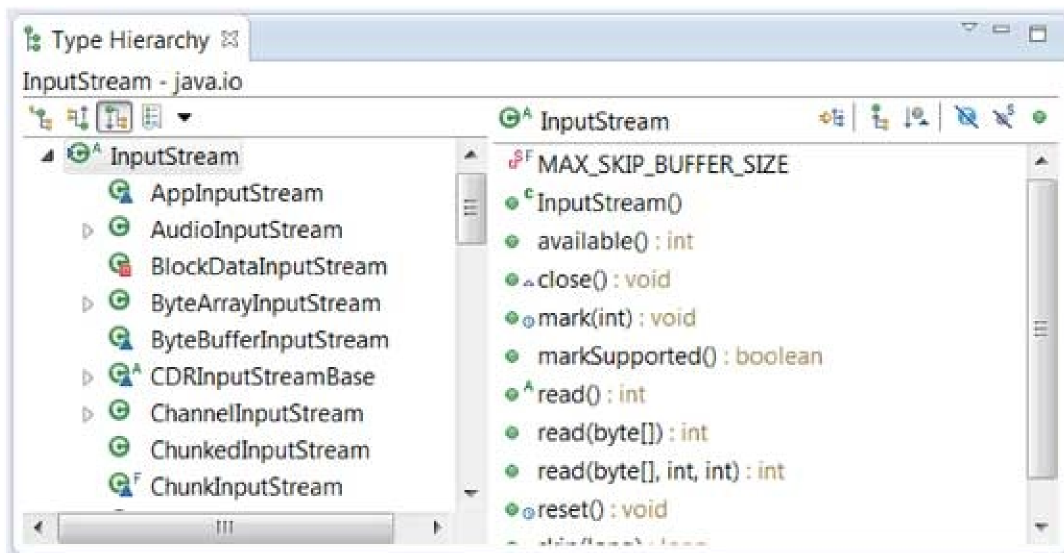
源代码位于 Java 主目录中的 src.zip 压缩文件中。

# Open Type in Hierarchy

Open Type in Hierarchy 菜单允许用户在 Type Hierarchy 视图中查看类的继承层次。



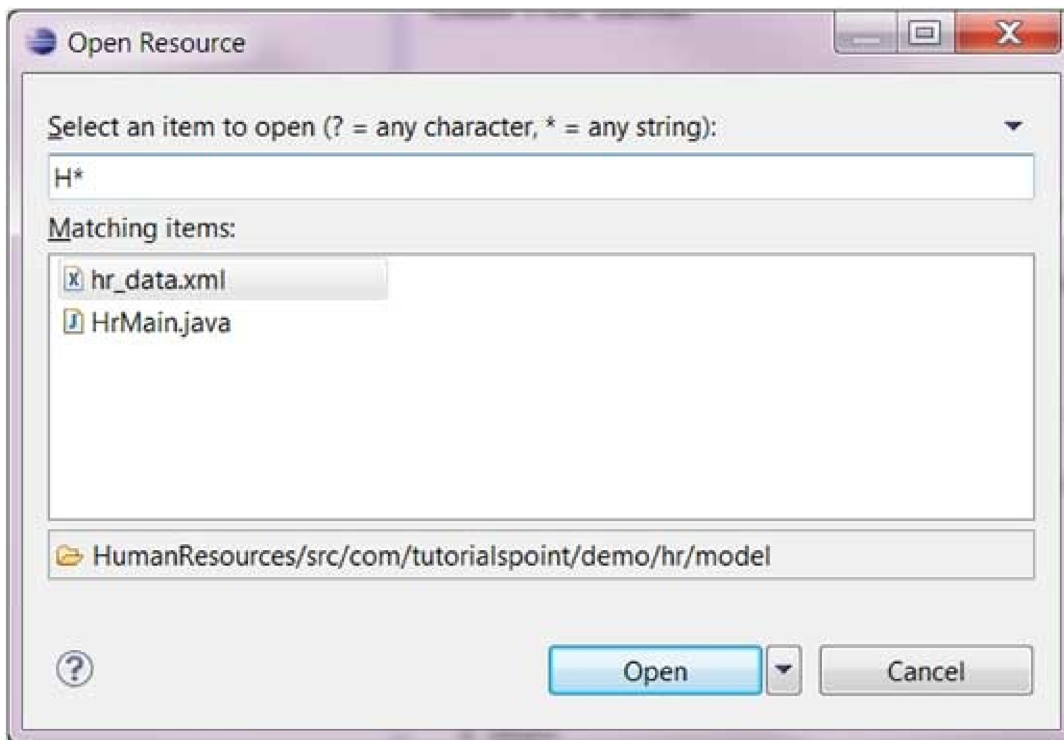

Type Hierarchy视图中选择指定的类就可以看到类的定义信息，包含对应的属性和方法：




# Open Resource

open resource(打开资源)菜单可用于查找工作空间中的文件。

'*' 号表示 0 个或多个字母，'?' 号表示单个字母可用于指定模式。对话框中将显示所有匹配的模式。




选择你要打开的文件并点击 OK 按钮。

# Eclipse 重构菜单

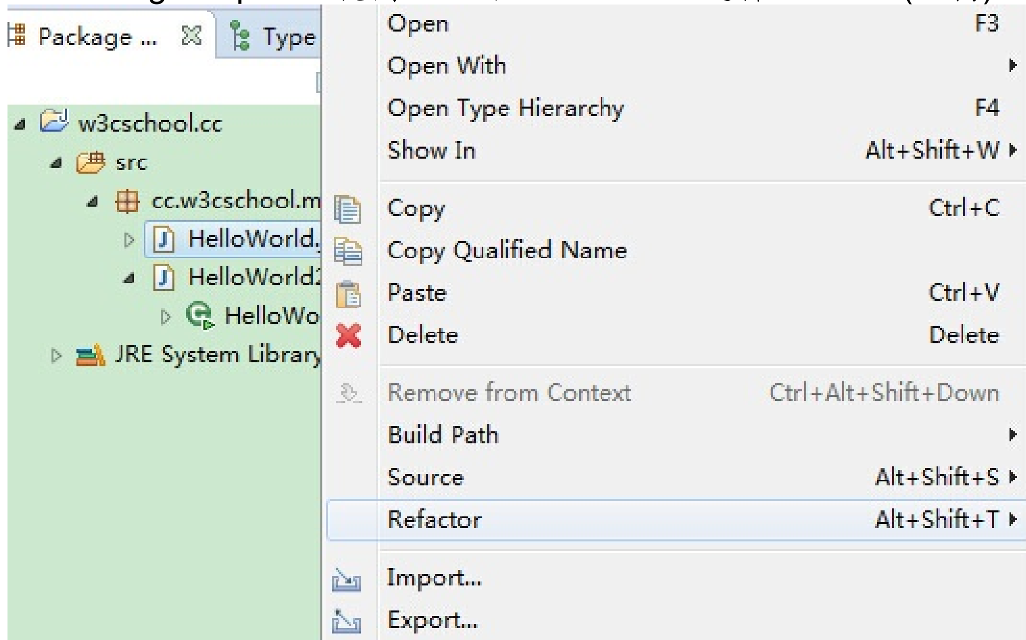
## 使用Eclipse重构

在项目开发中我们经常需要修改类名，但如果其他类依赖该类时，我们就需要花很多时间去修改类名。

但 Eclipse 重构功能可以自动检测类的依赖关系并修改类名，帮我们节省了很多时间。

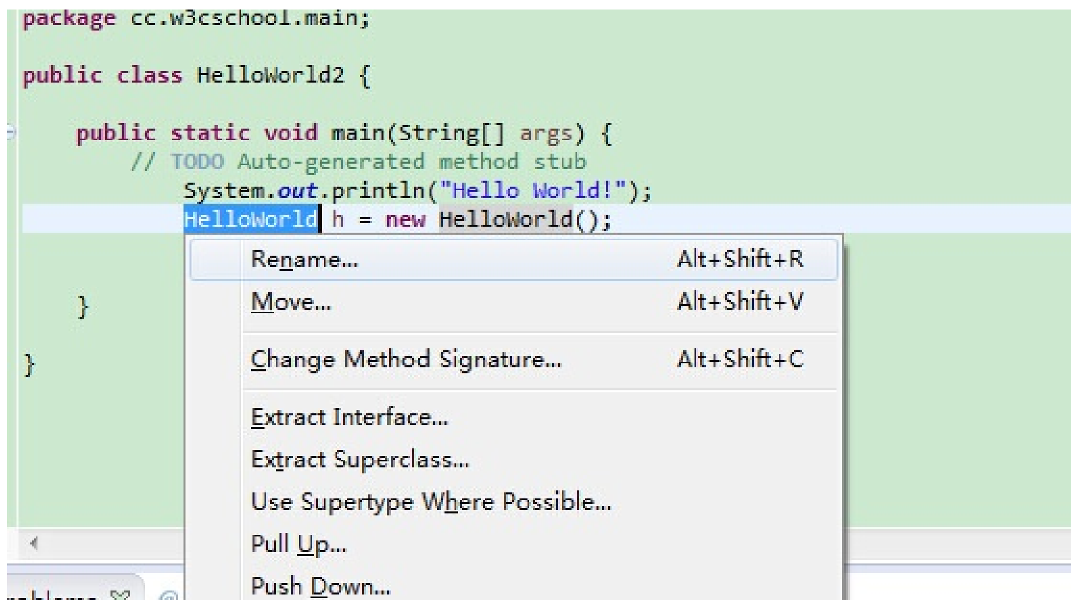
可用过以下方式打开重构菜单：

- 在 Package Explorer 视图中右击 Java 元素并选择Refactor(重构)菜单项

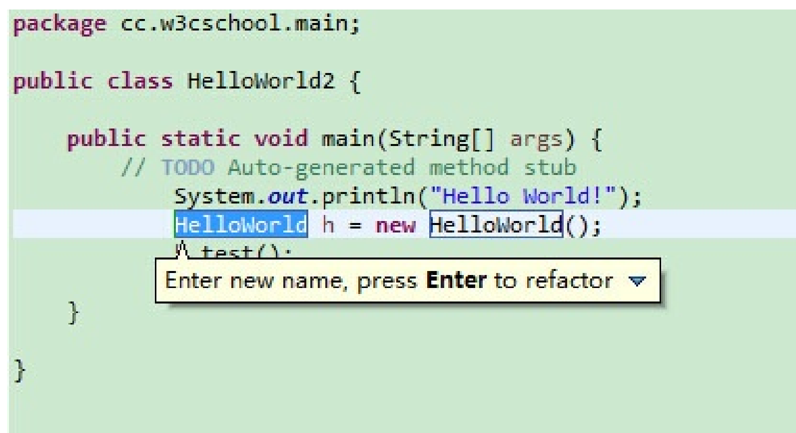



- 在 Java 编辑器中鼠标右击 Java 元素并选择Refactor(重构)菜单项
- 在 Package Explorer 视图中选择 Java 元素并按下 Shift + Alt + T

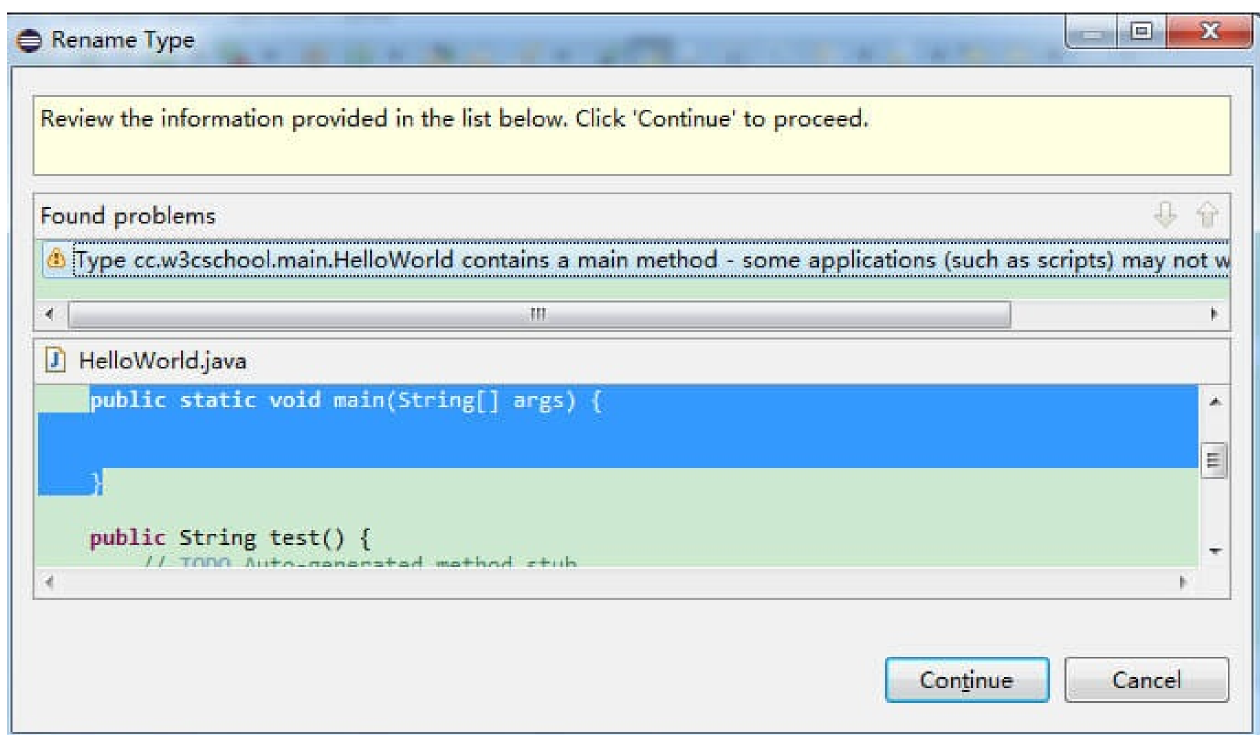
下图中我们在Java 编辑器中选中了 HelloWorld 类：



在选择 Rename 后会提示输入新的类名并按回车结束修改：



在修改完成按下回车键后，会弹出将将会修改的类 ：



你只需点击 Continue 按钮即可完成操作。

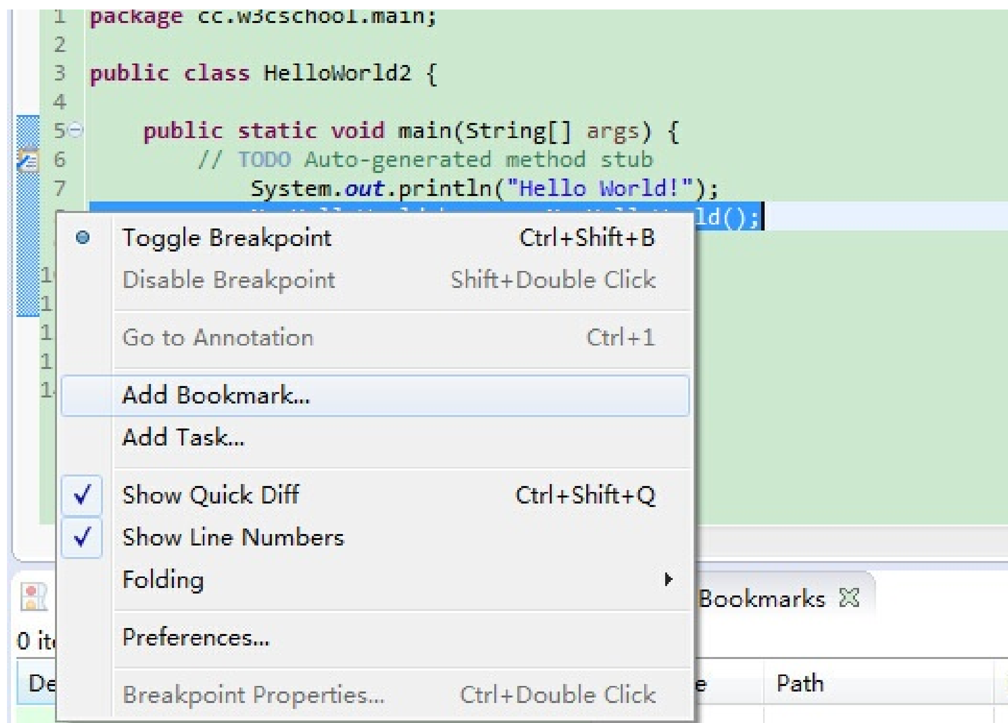
# Eclipse 添加书签

## 关于书签

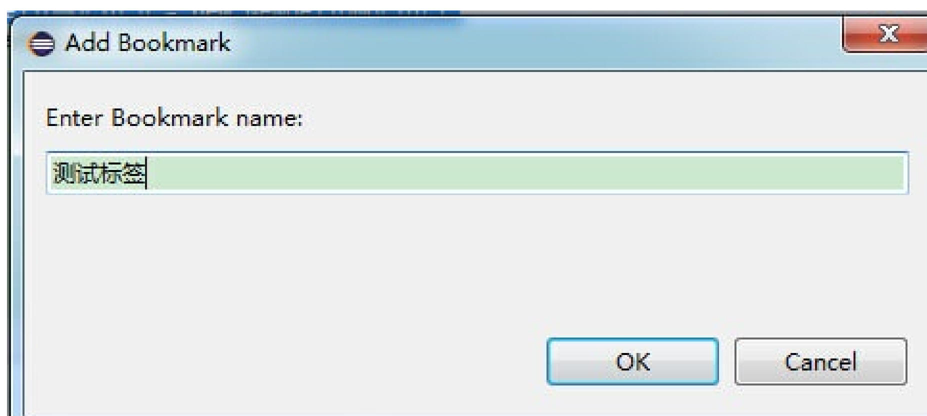
Eclipse 中可以在编辑器的任意一行添加书签。 您可以使用书签作为提示信息，或者使用书签快速定位到文件中的指定的行。

## 添加书签

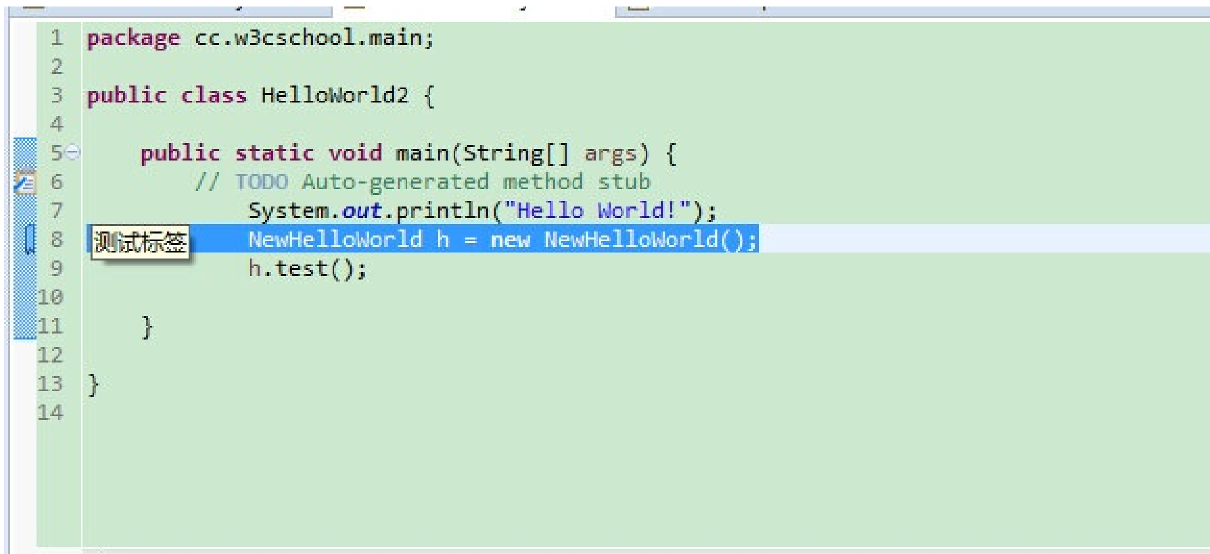
如果你想设置书签，你只需要在垂直标尺上右击鼠标并选择能 "Add Bookmark" 即可。




在弹出的对话框中输入书签名。



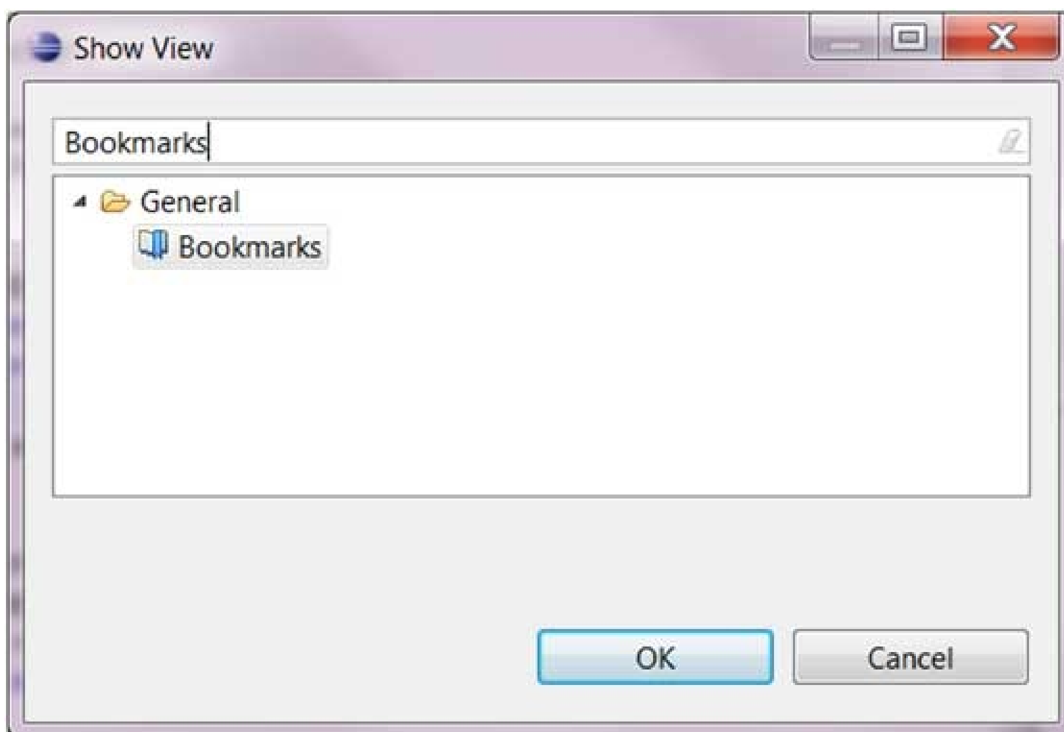
垂直标尺上就会出现一个书签的按钮，当然你也可以在 Bookmarks 视图中查看到书签列表。




## 打开 Bookmarks(书签) 视图

打开 Bookmarks 视图的方法为:

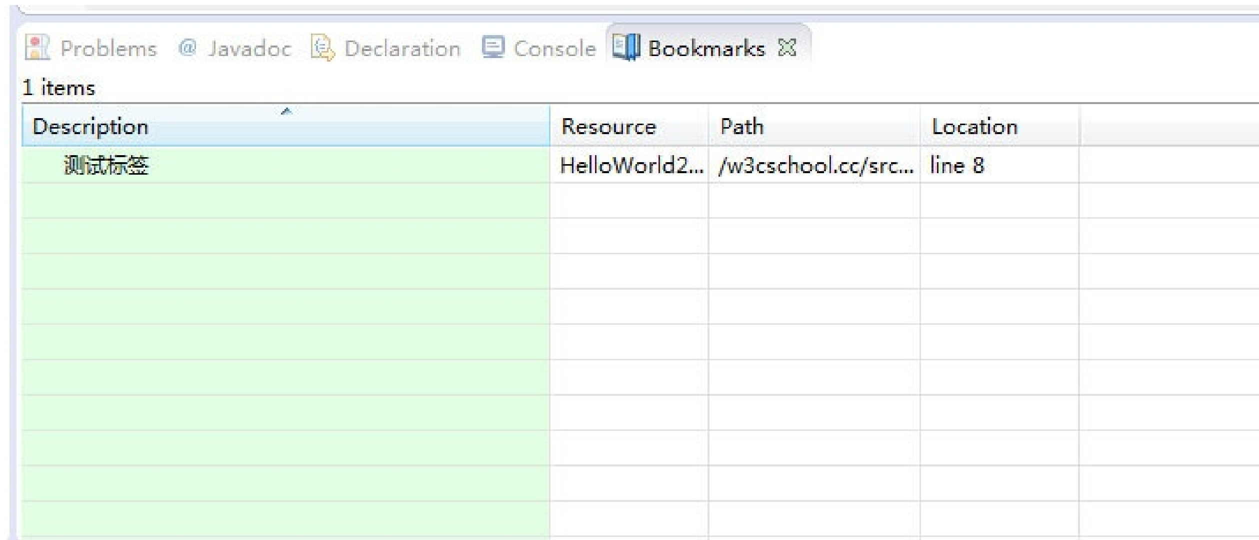
- 点击 Window 菜单选择 Show View > Other
- 在搜索输入框中输入 Bookmark
- 在 General 下选择 Bookmarks




- 点击 OK 按钮

# 使用 Bookmarks(书签) 视图

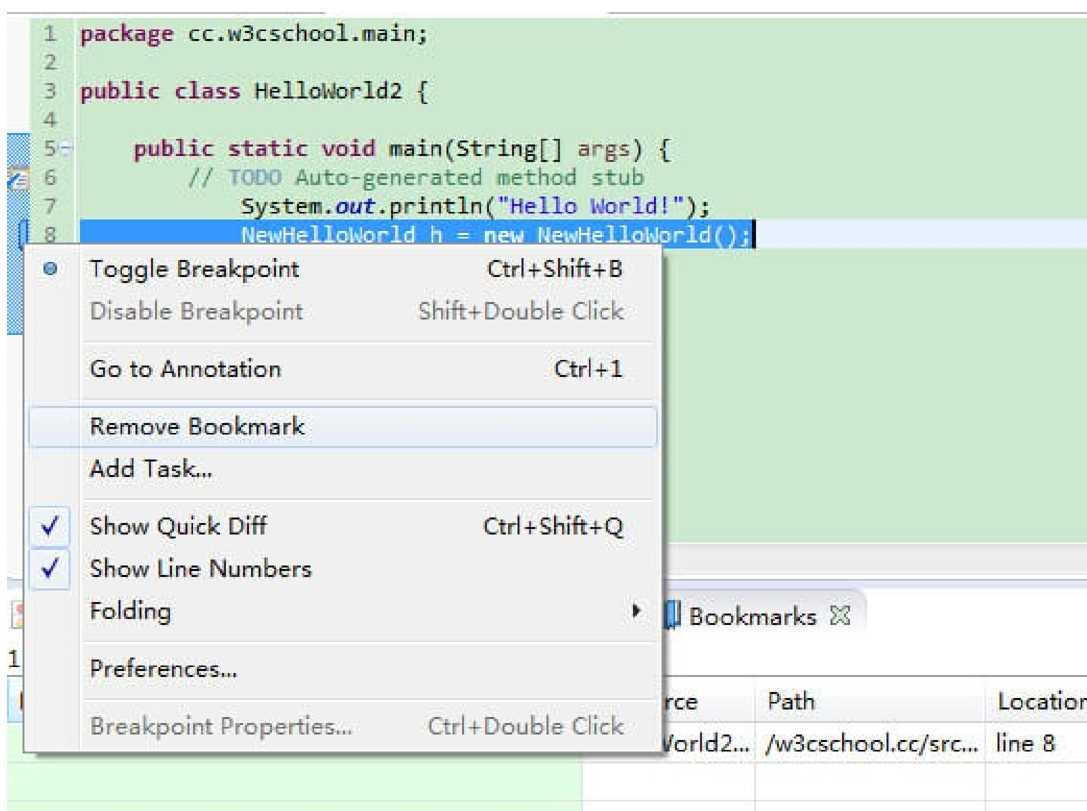
Bookmarks 视图如下：



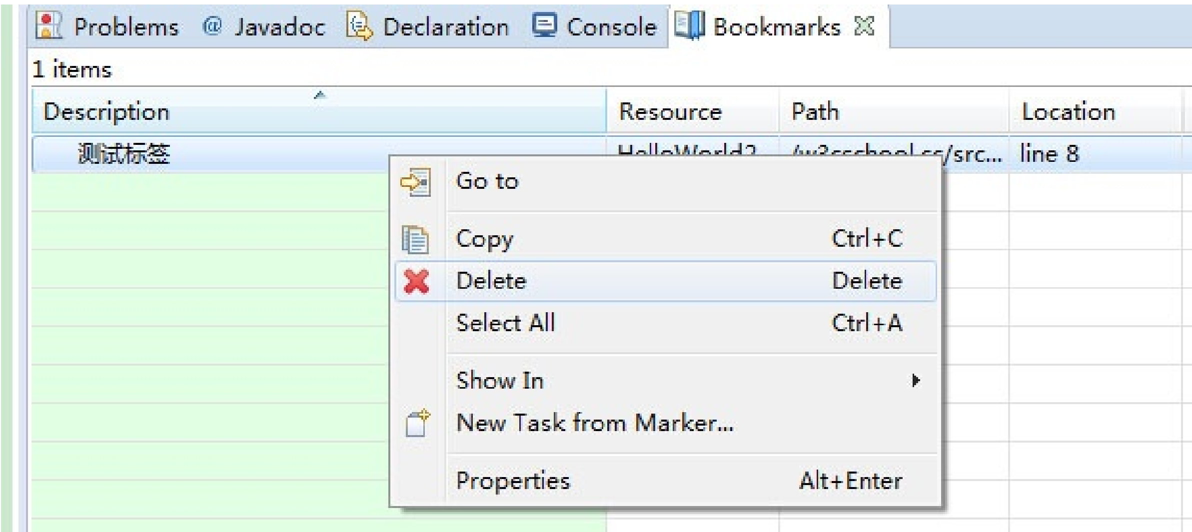

你可以在 Bookmarks 视图中双击书签或者鼠标右击书签选择"Go to"菜单来快速定位书签所在的位置。

# 删除Bookmarks(书签)

你可以在垂直书签上右击编辑并选择 Remove Bookmark 来删除书签：




或者在 Bookmarks 视图视图中右击书签并选择"Delete"菜单项来删除书签：

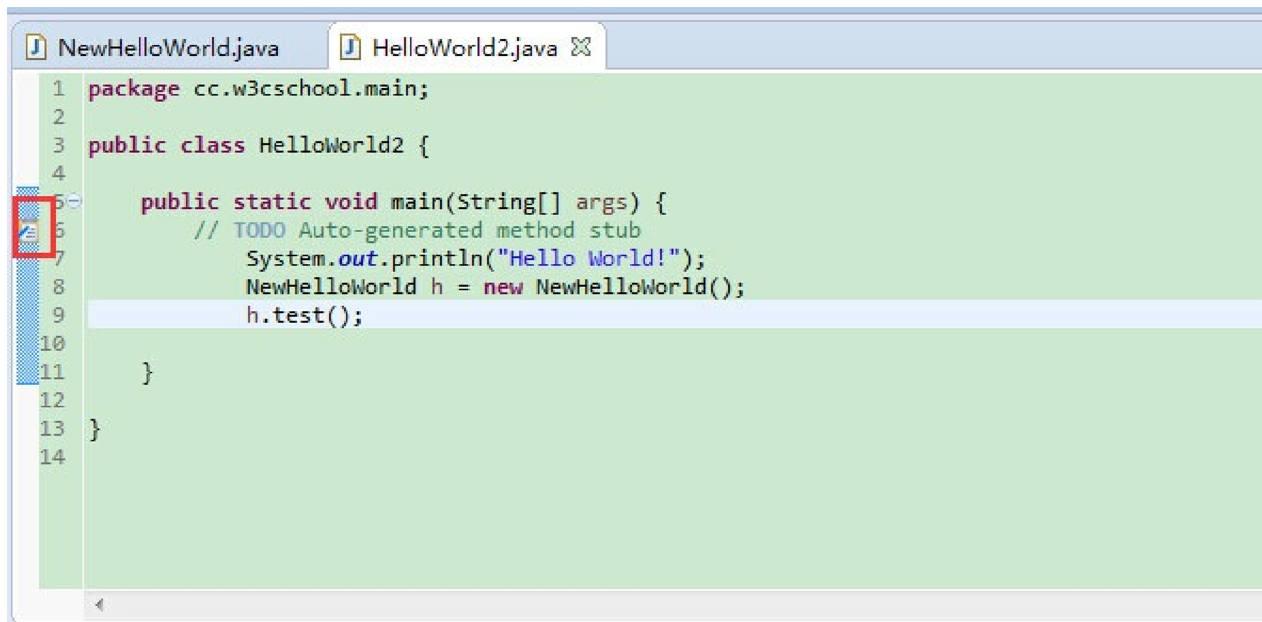

Problems
Javadoc
Declaration
Console
Bookmarks
1 items
Description
Resource
Path
Location
测试标签
line 8
Go to
Copy Ctrl+C
Delete Delete
Select All Ctrl+A
Show In
New Task from Marker...
Properties Alt+Enter

# Eclipse 任务管理

## 管理任务

在Eclipse中用TODO标签管理任务，利用这个功能可以方便地将项目中一些需要处理的任务记录下来。

我们可以在 Java 代码中的注释添加 TODO 单词来标记一个任务，任务可以通过 Tasks(任务) 视图查看。




在Eclipse中我们可以通过鼠标右击垂直标尺并选择 Add Task 菜单来添加任务，在弹出的对话框中输入任务描述信息：

NewHelloWorld.java | HelloWorld2.java

```
1  package cc.w3cschool.main;
2
3  public class HelloWorld2 {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          System.out.println("Hello World!");
8          NewHelloWorld h = new NewHelloWorld();
```

Toggle Breakpoint Ctrl+Shift+B
Disable Breakpoint Shift+Double Click
Go to Annotation Ctrl+1
Add Bookmark...
Add Task...
Show Quick Diff Ctrl+Shift+Q

Properties

Description: 测试函数还未完成

Priority: Normal Completed

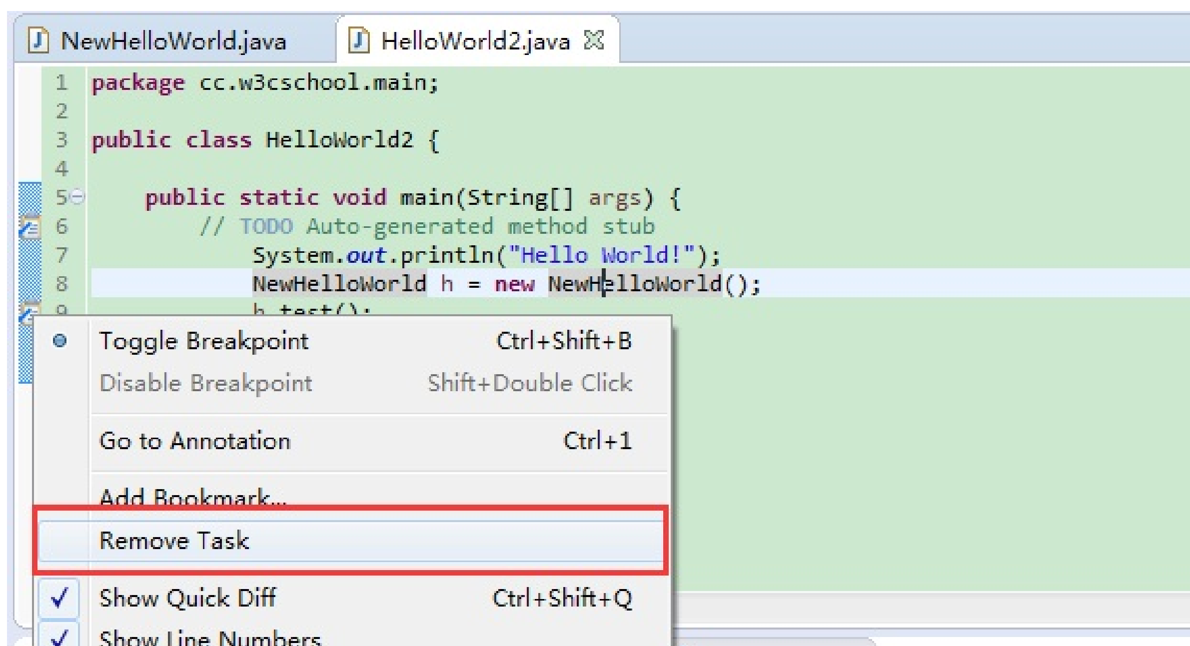
On element: HelloWorld2.java

In folder: w3cschool.cc/src/cc/w3cschool/main

Location: line 9

OK Cancel

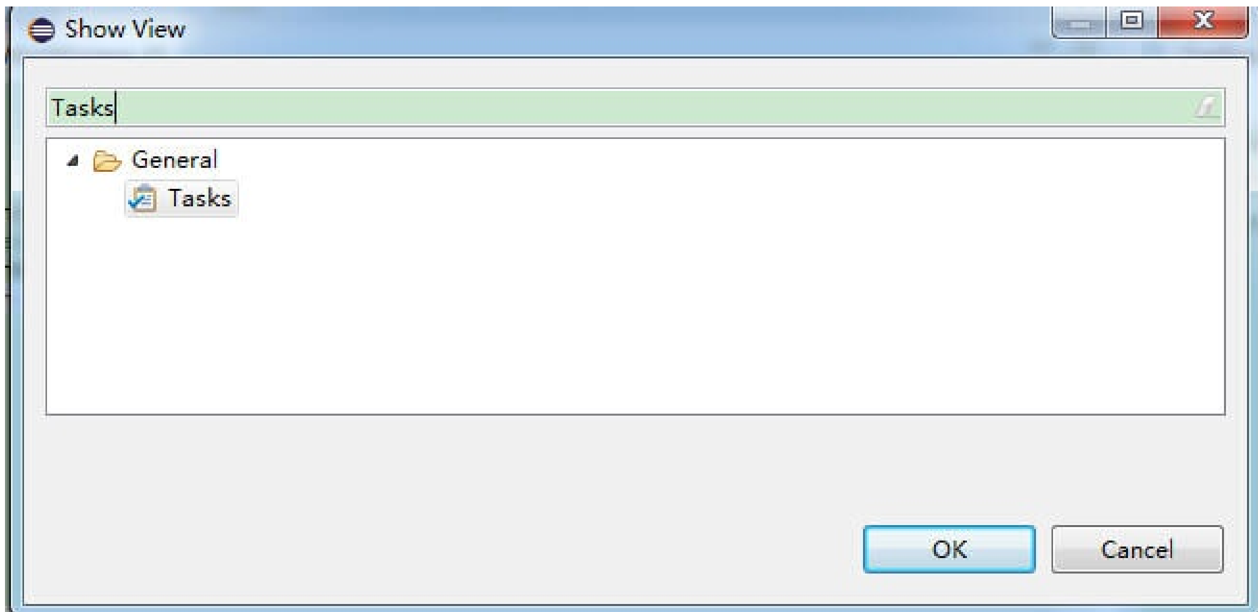
如果需要删除任务，只需右击任务图标选择 Remove Task 菜单项即可：



# 打开 Task(任务) 视图

打开 Task(任务) 视图的方法为：

- 在 Window 菜单中选择 Show View > Other
- 在搜索框中输入 Tasks
- 在 General 下选择 Tasks




- 最后点击 OK 按钮

# 使用 Task(任务) 视图

Task(任务) 视图中显示了项目中所有待完成的任务：

Problems @ Javadoc Declaration Console Tasks

3 items

| | ! | Description | Resource | Path | Location | Type |
|---|---|---|---|---|---|---|
| ☐ | | 测试函数还未完成 | HelloWorld2... | /w3cschool.cc/src... | line 9 | Task |
| | | TODO Auto-generated method stub | HelloWorld2... | /w3cschool.cc/src... | line 6 | Java Task |
| | | TODO Auto-generated method stub | NewHelloW... | /w3cschool.cc/src... | line 11 | Java Task |

Task(任务) 视图中还能进行以下操作：

- 修改任务右下角
- 标记任务已完成
- 删除任务或删除所有已完成任务

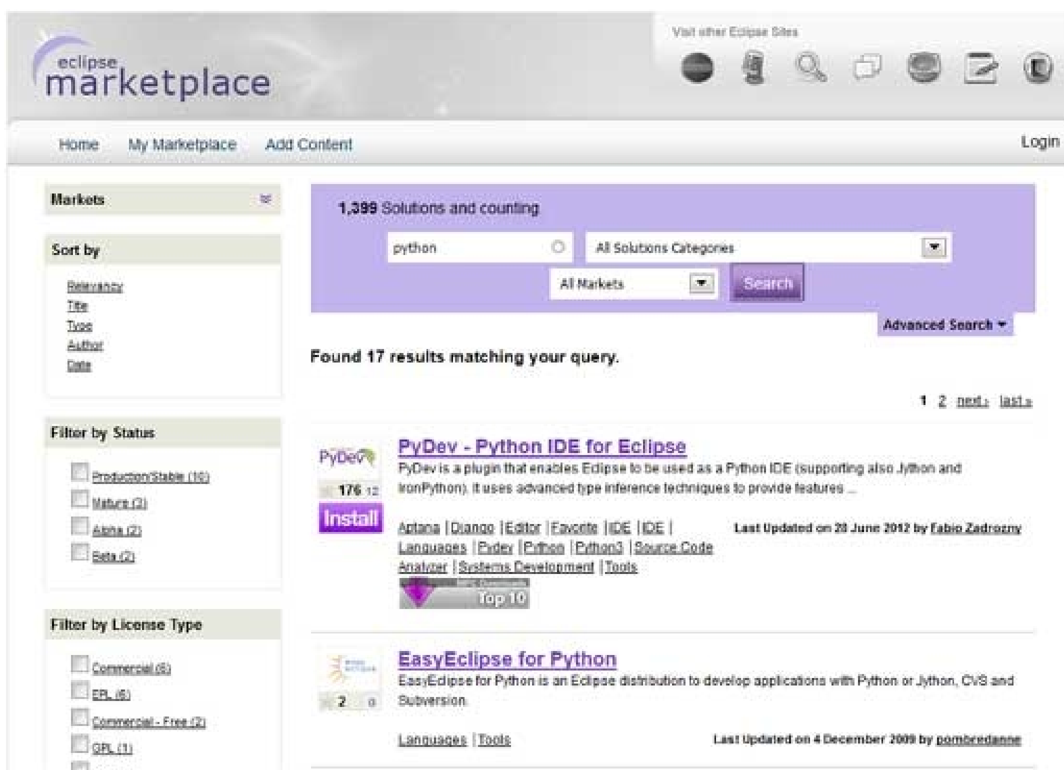
# Eclipse 安装插件

## 查找和安装插件

Eclipse作为一个集成的IDE开发工具，为我们的软件开发提供了便利，eclipse除了自带的强大功能外，还支持功能丰富的插件。

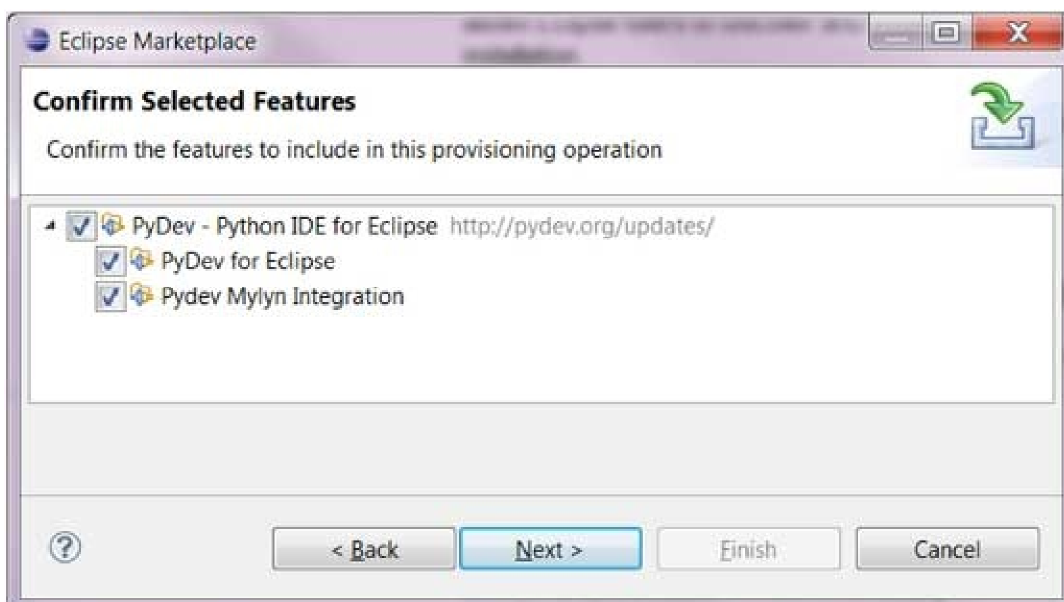
我们可以通过Eclipse官方市场 (http://marketplace.eclipse.org/)找到并下载我们需要的插件。

例如我们可以查找支持 Python IDE 的插件，如下图所示：

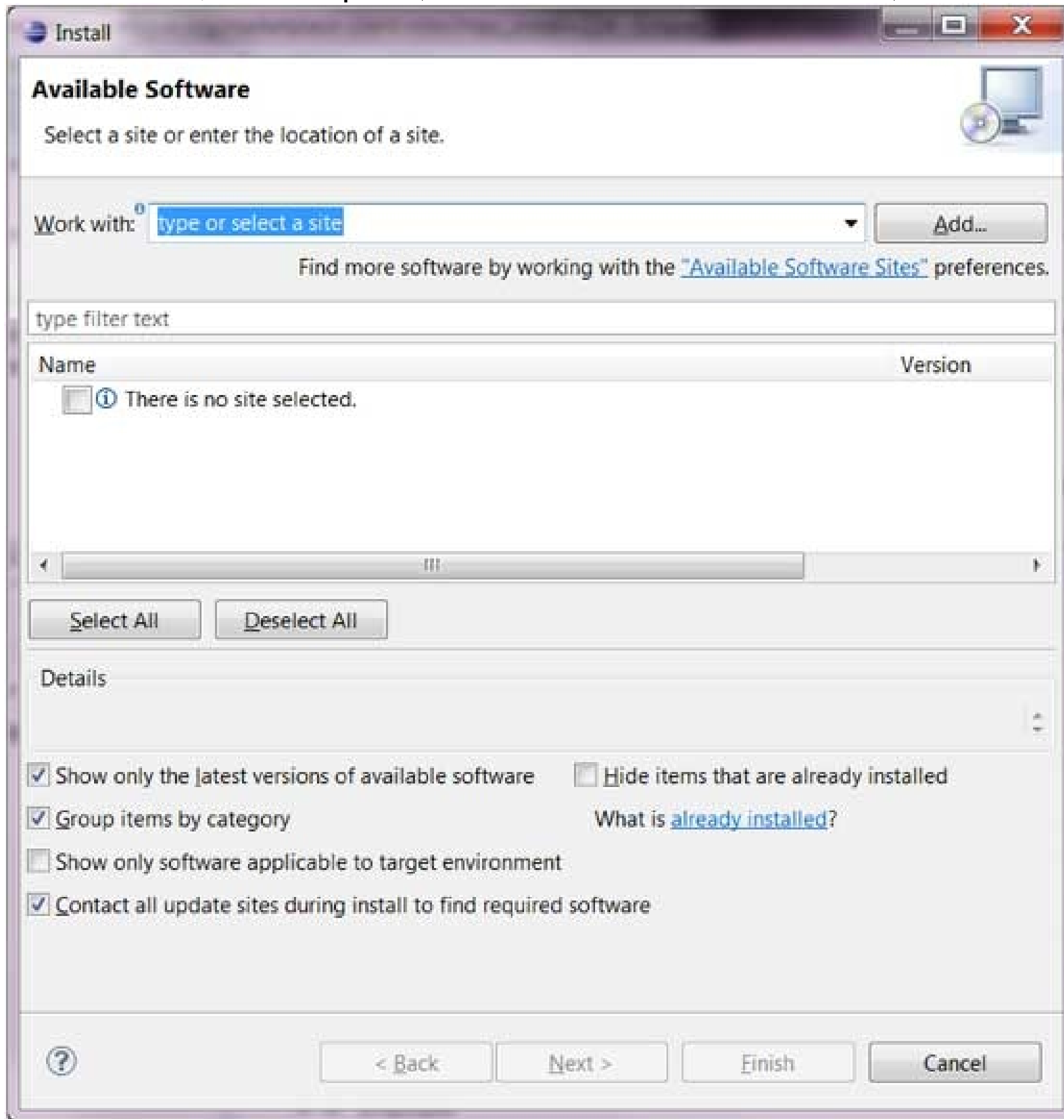



在 Eclipse IDE 中我们也可以通过点击 Help 菜单中的 Eclipse Marketplace（Eclipse 超市）选项来查找插件：

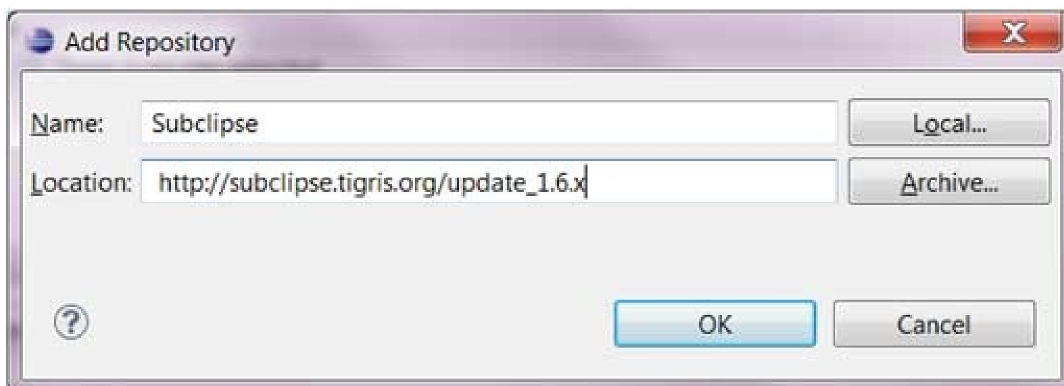
上图中我们选择 PyDev 让 Eclipse 支持 Python 开发，我们只需要点击 Install 按钮即可。 以下对话框为选择安装的插件。



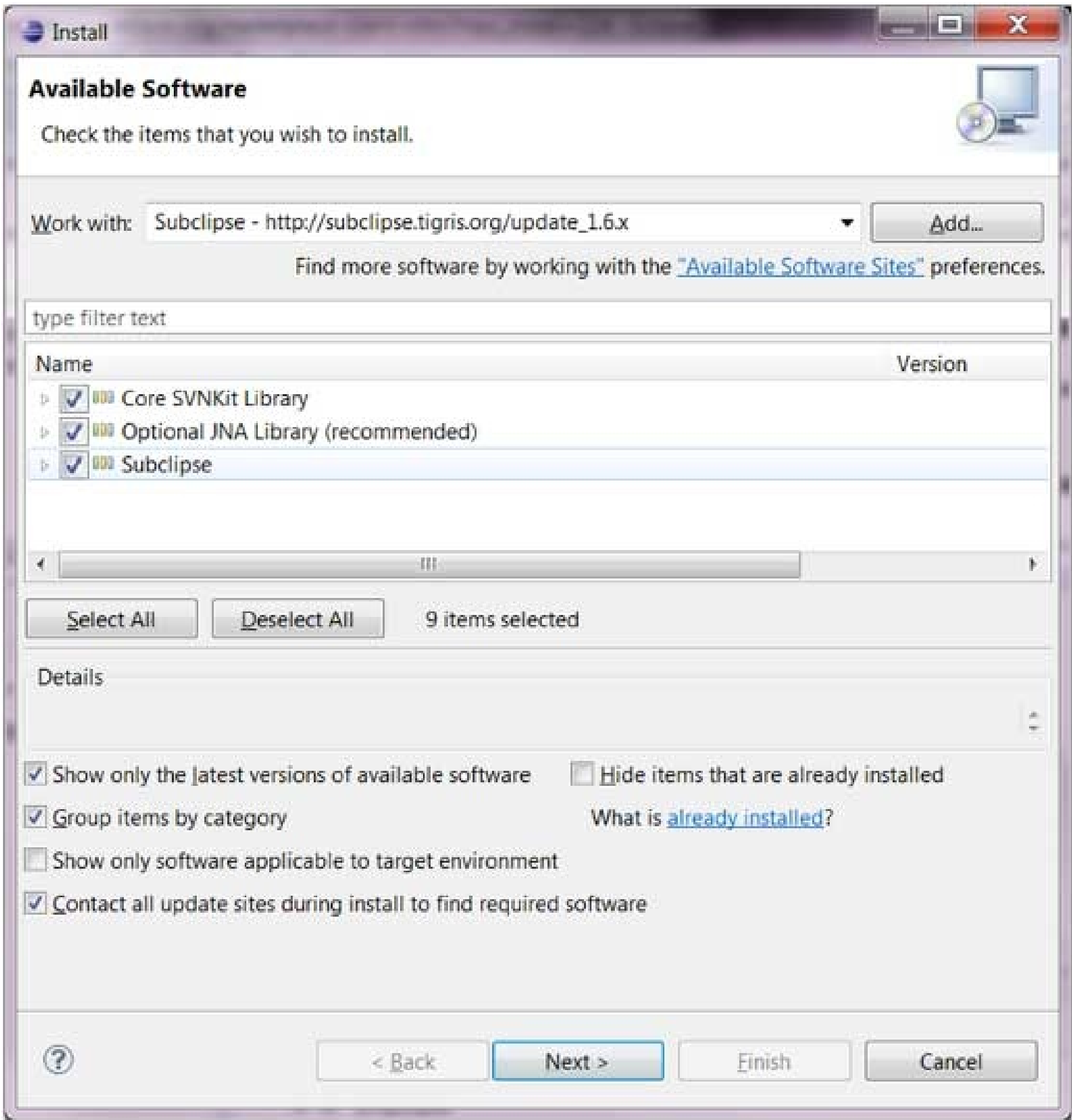
你也可以通过点击 Help 菜单上的 Install New Software 菜单项来安装插件：




这种方式我们需要知道插件远程的安装地址，你可以通过点击 Add 按钮来提交 URL。




安装的对话框中列出了远程可安装的插件列表：


Install
Available Software
Check the items that you wish to install.
Work with: Subclipse - http://subclipse.tigris.org/update_1.6.x
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
Core SVNKit Library
Optional JNA Library (recommended)
Subclipse
Select All
Deselect All
9 items selected
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel

# Eclipse 代码模板

## 使用代码模板

Eclipse 提供了通过定义和使用代码模板来提高工作效率与代码可预测性的能力。

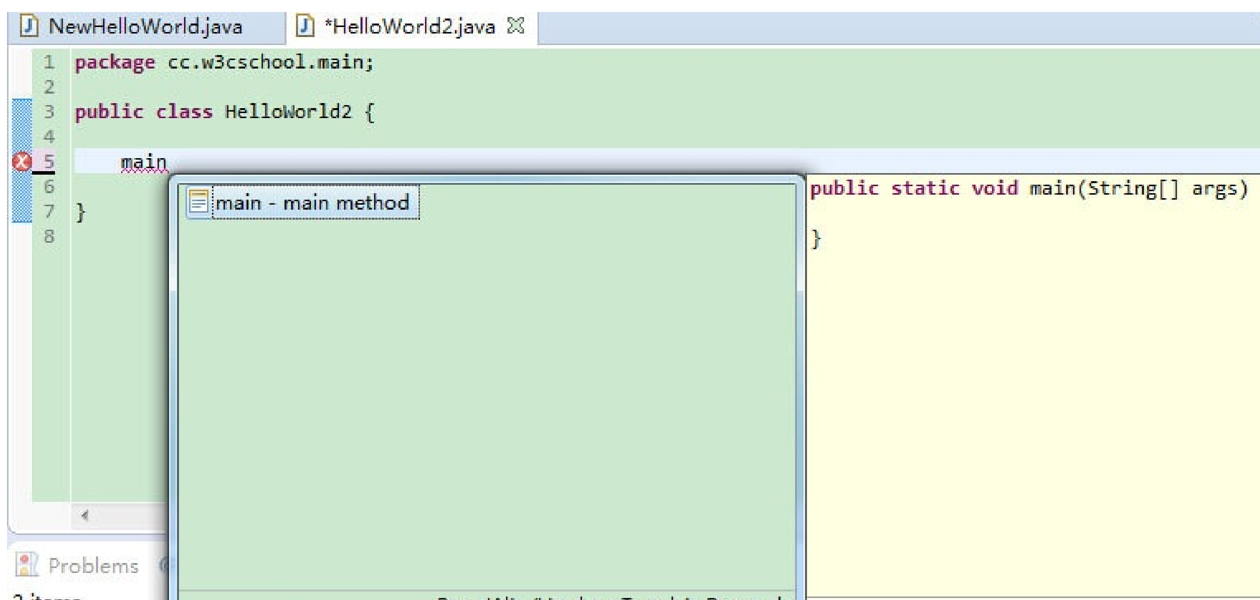
我们在开发 Java 程序过程中经常需要编写 main 方法：

```
public static void main(String[]args) {

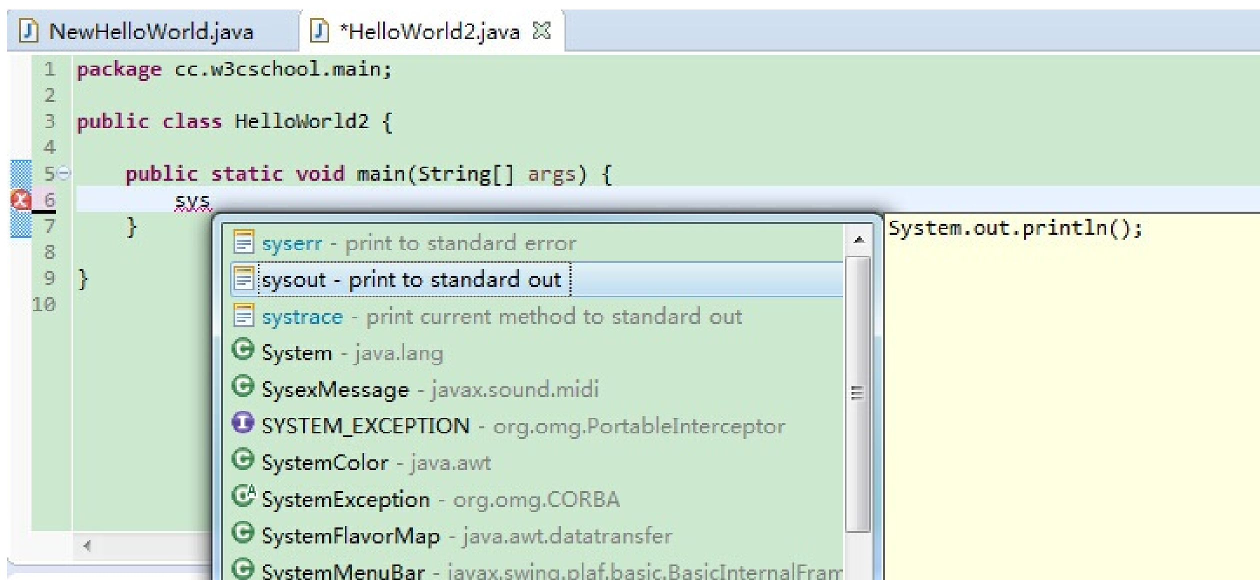
}
```

如果我们一个字母一个字母去编写，将是一个重复而又毫无意义的事情，这是我们就可以使用 Eclipse 代码模板来快速完成这些工作。

我们只需在类体中键入main，然后使用Eclipse的代码提示快捷键(默认为Alt+/)，回车后，就可以看到Eclipse自动帮我们完成了main函数的完整定义：

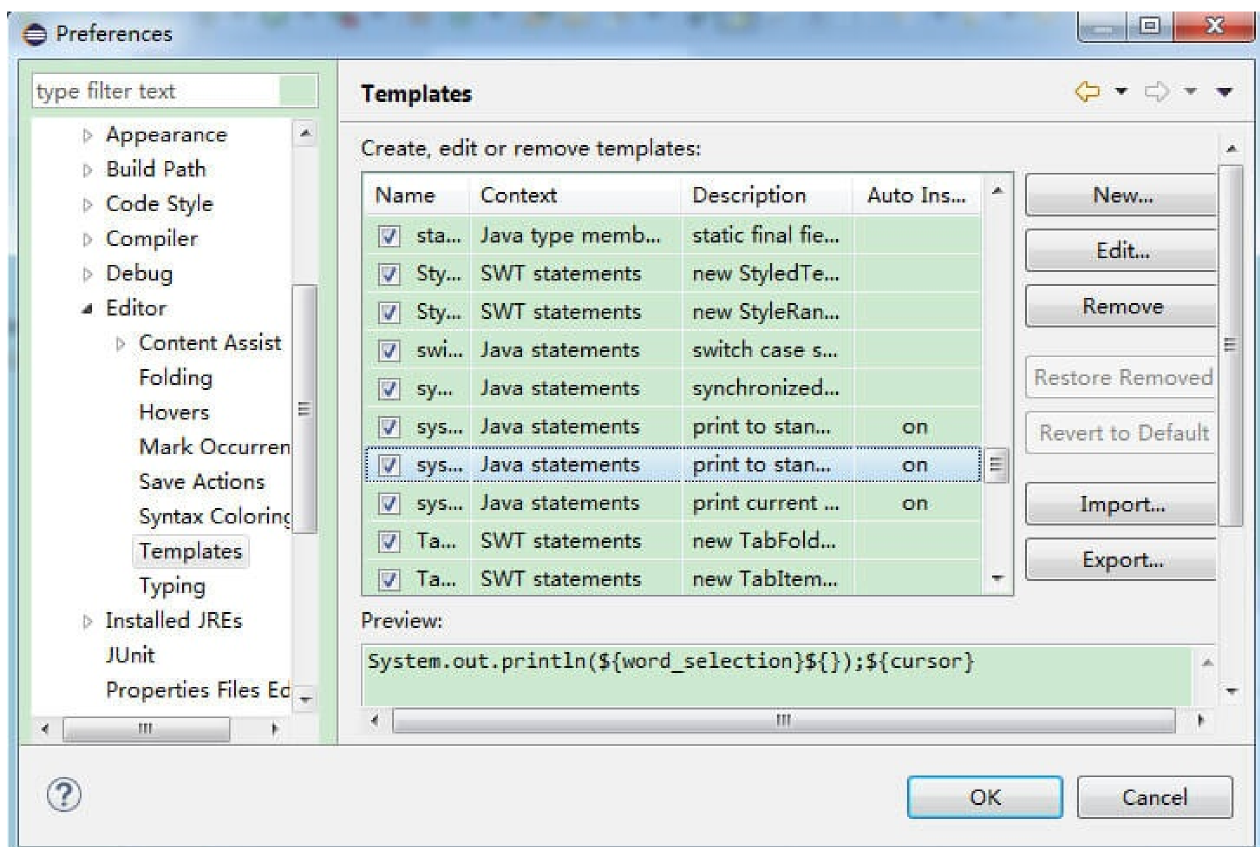



如果我们要使用 System.out.println()，我们只需要输入 syso 然后按下 Alt+/ 即可：

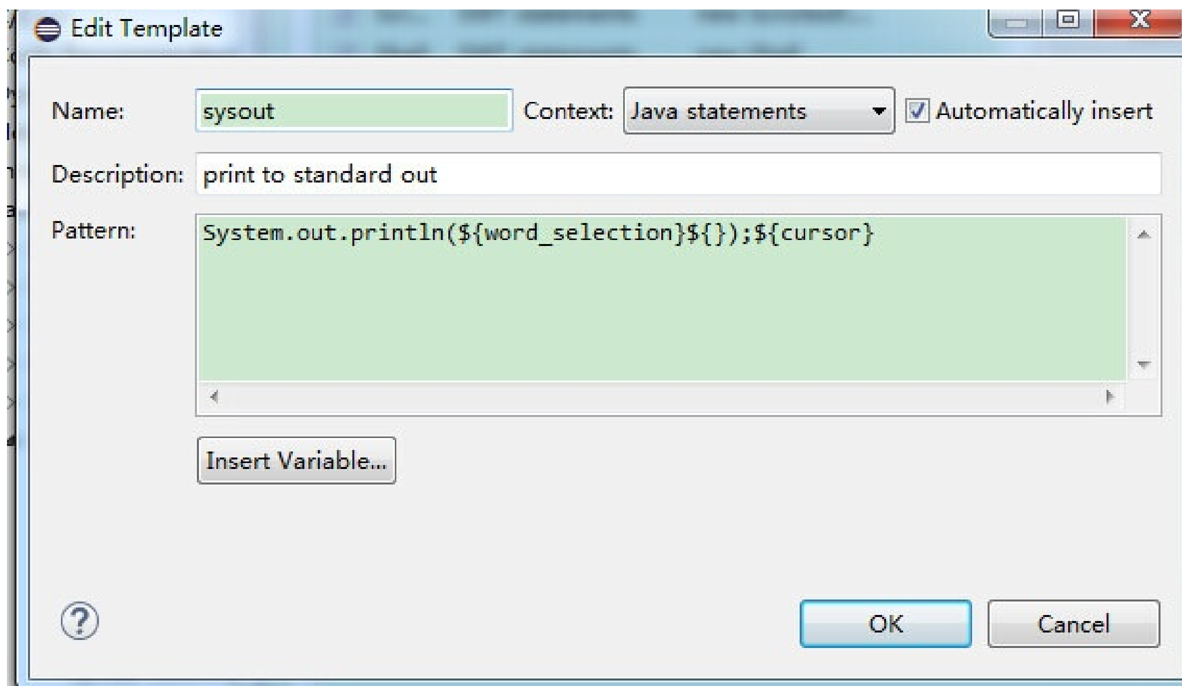


# 自定义代码模板

Eclipse 还提供了非常多的代码模板，我们可以通过 Windows->Preferences->Java->Editor->Templates (你可以在搜索框中输入Templates查找)看到所有已定义的代码模板列表。



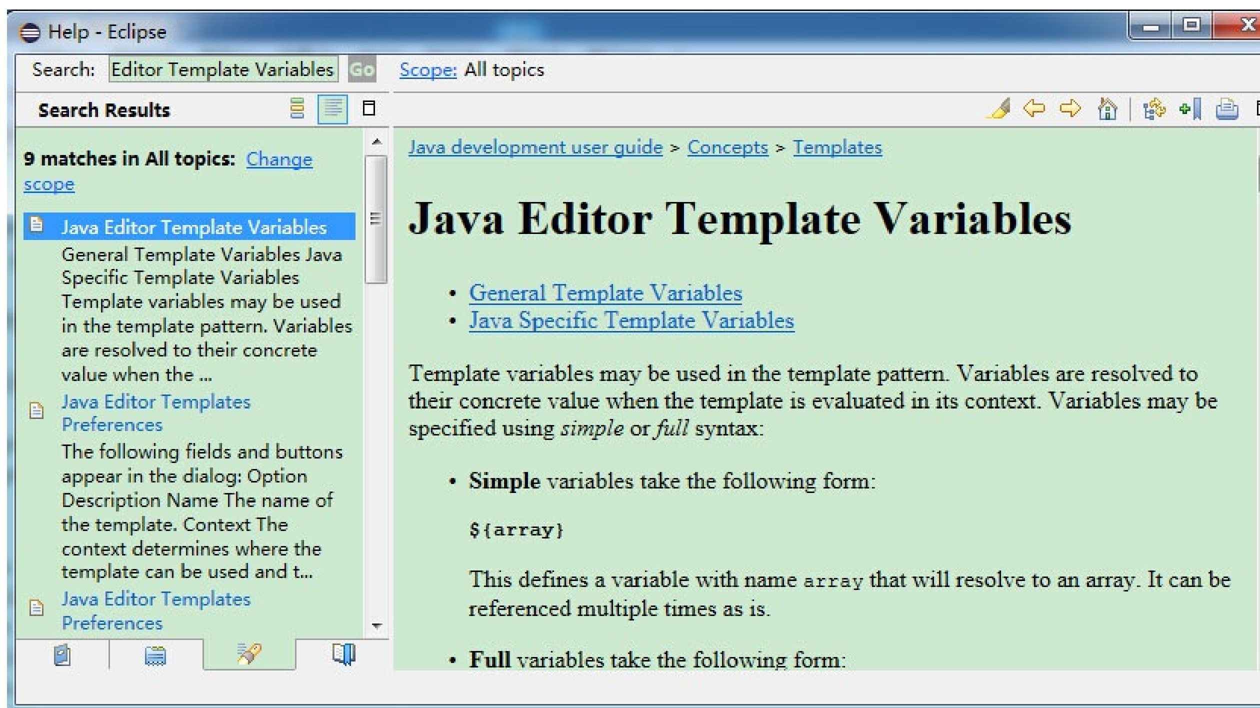
我们在弹窗口选中 sysout 模板并点击右侧Edit，显示如下：




编辑面板是核心关注对象，因为一切东西都在这里面配置。先来熟悉下这个面板中关键的五项分别是什么。

- **Name**：名称，其实就是以后可以用到的代码缩写
- **Context**：模板上下文，指定该代码模板在什么地方才能生效，对于Java至少包含这么四个：
    1. Java type members，模板对应的代码是类成员，psvm模板严格来说应该选择这个
    2. Java statements，模板对应的代码是语句块
    3. Java，最通用的，只要是Java代码就行
    4. Java doc，顾名思义了
- 模板变量：eclipse已经预置了一些模板变量（点Insert Varibles可以看到所有预置变量），如：
    1. ${cursor}是表示光标
    2. ${date}表示当前日期字符串
    3. ${time}表示当前时间字符串
    4. ${line_selection}让当前行被选中
    5. ${word_selection}让当前单词被选中当然我们也可以定义自己的模板变量，比如我定义一个 ${myTemplateVarible}，那么对应代码显示的就是myTemplateVarible。
- **Pattern**：代码模板对应的模式，按照你希望代码的格式逐个输入即可

更多自定义代码模板的内容你可以通过点击 Help 菜单中的 Help Contents 选项，在弹出的对话框的搜索栏上输入 "Java Editor Template Variables" 选择 Java Editor Template Variables 查看具体的文档描述：

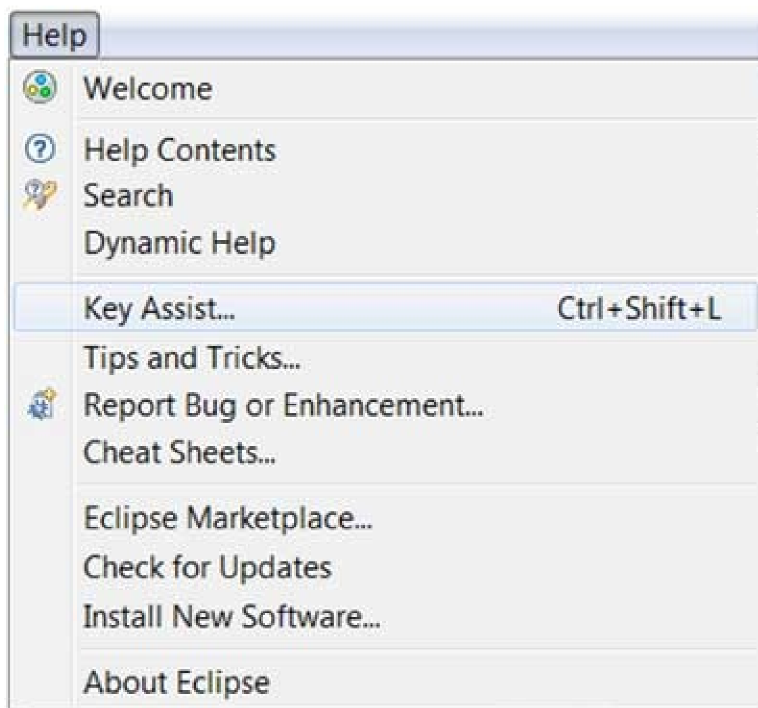

Help - Eclipse
Search: Editor Template Variables Go
Scope: All topics
Search Results
9 matches in All topics: Change scope
Java Editor Template Variables
General Template Variables Java Specific Template Variables Template variables may be used in the template pattern. Variables are resolved to their concrete value when the ...
Java Editor Templates Preferences
The following fields and buttons appear in the dialog: Option Description Name The name of the template. Context The context determines where the template can be used and t...
Java Editor Templates Preferences
Java development user guide > Concepts > Templates
Java Editor Template Variables
General Template Variables
Java Specific Template Variables
Template variables may be used in the template pattern. Variables are resolved to their concrete value when the template is evaluated in its context. Variables may be specified using simple or full syntax:
Simple variables take the following form:
${array}
This defines a variable with name array that will resolve to an array. It can be referenced multiple times as is.
Full variables take the following form:

# Eclipse 快捷键

## 关于快捷键

Eclipse 的很多操作都提供了快捷键功能，我们可以通过键盘就能很好的控制 Eclipse 各个功能：

- 使用快捷键关联菜单或菜单项
- 使用快捷键关联对话窗口或视图或编辑器
- 使用快捷键关联工具条上的功能按钮




Eclipse 快捷键列表可通过快捷键 Ctrl + Shift + L 打开 。

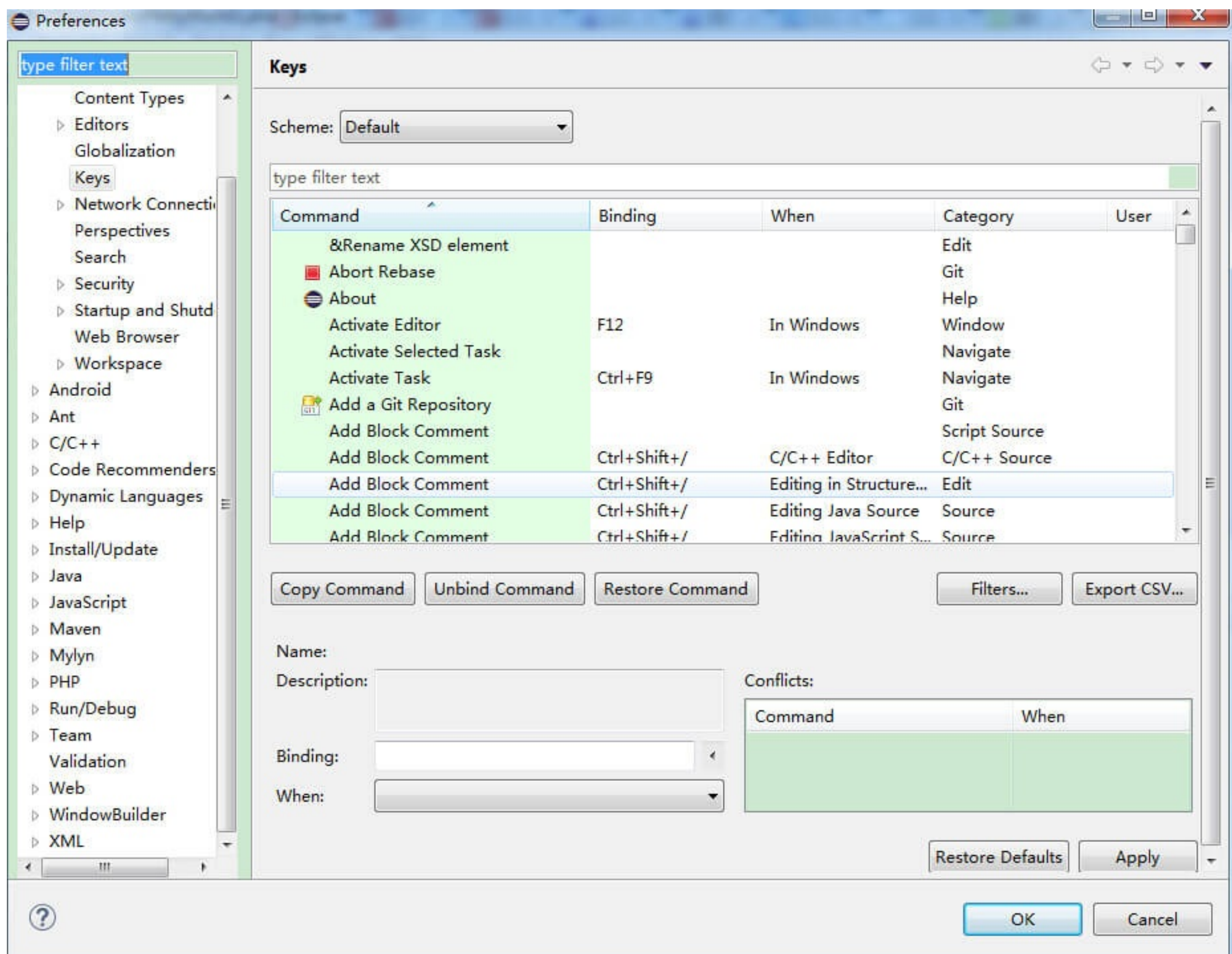
| Activate Editor | F12 |
| --- | --- |
| Activate Task | Ctrl+F9 |
| Add Javadoc Comment | Alt+Shift+J |
| All Instances | Ctrl+Shift+N |
| Backward History | Alt+Left |
| Build All | Ctrl+B |
| Change Method Signature | Alt+Shift+C |
| Close | Ctrl+F4 |
| Close All | Ctrl+Shift+F4 |
| Collapse All | Ctrl+Shift+Numpad_Divide |
| Commit... | Ctrl+# |
| Content Assist | Ctrl+Space |
| Context Information | Ctrl+Shift+Space |
| Copy | Ctrl+Insert |
| Cut | Shift+Delete |
| Deactivate Task | Ctrl+Shift+F9 |
| Debug | F11 |
| Debug Ant Build | Alt+Shift+D, Q |
| Debug JUnit Test | Alt+Shift+D, T |
| Debug Java Applet | Alt+Shift+D, A |
| Debug Java Application | Alt+Shift+D, J |

Press "Ctrl+Shift+L" to open the preference page.

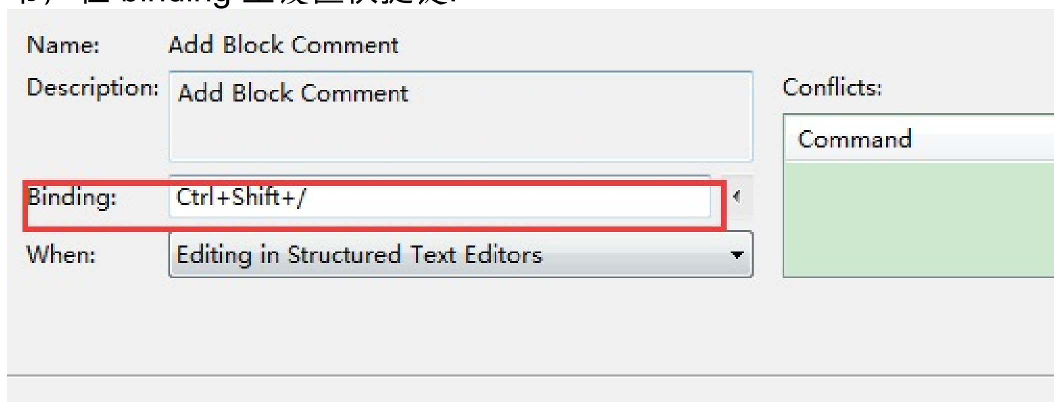
## 设置快捷键

Eclipse 系统提供的快捷键有时比较难记住，甚至根本没有提供快捷键时，就需要自己手动设置快捷键。

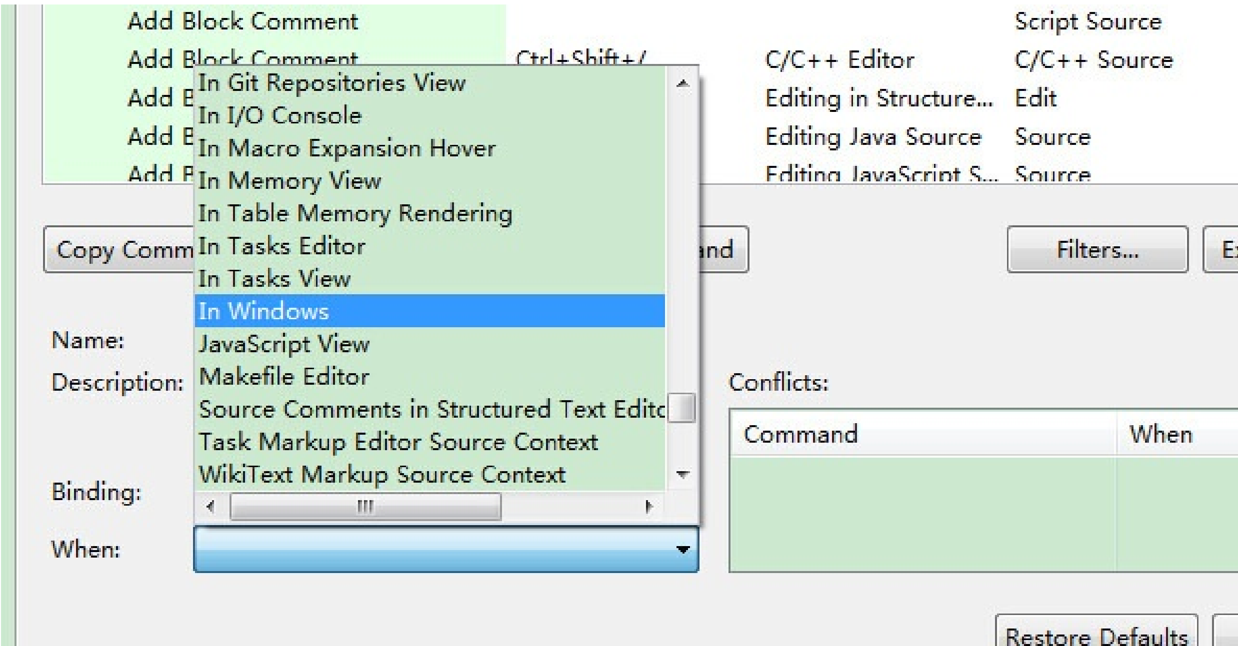
我们可以通过点击window->preferences->general->keys（或直接搜索keys），进入快捷键管理界面：




在这里可以查找所有功能的快捷键，需要修改或新增时，点击需要修改或新增的命令，在 binding 里设置快捷键:




设置完快捷键后，还需要设置在什么时候可以使用该快捷键，eclipse提供各种场景供选择，一般选择In Windows(即在eclipse窗口激活状态)即可。




完成以上操作，点击 OK 按钮即完成设置。

# Eclipse 常用快捷键

| 快捷键 | 描述 |
|---|---|
| 编辑 | |
| Ctrl+1 | 快速修复（最经典的快捷键,就不用多说了，可以解决很多问题，比如import类、try catch包围等） |
| Ctrl+Shift+F | 格式化当前代码 |
| Ctrl+Shift+M | 添加类的import导入 |
| Ctrl+Shift+O | 组织类的import导入（既有Ctrl+Shift+M的作用，又可以帮你去除没用的导入，很有用） |
| Ctrl+Y | 重做（与撤销Ctrl+Z相反） |
| Alt+/ | 内容辅助（帮你省了多少次键盘敲打，太常用了） |
| Ctrl+D | 删除当前行或者多行 |
| Alt+↓ | 当前行和下面一行交互位置（特别实用,可以省去先剪切,再粘贴了） |
| Alt+↑ | 当前行和上面一行交互位置（同上） |
| Ctrl+Alt+↓ | 复制当前行到下一行（复制增加） |
| Ctrl+Alt+↑ | 复制当前行到上一行（复制增加） |
| Shift+Enter | 在当前行的下一行插入空行（这时鼠标可以在当前行的任一位置,不一定是最后） |

| | |
|---|---|
| Ctrl+/ | 注释当前行,再按则取消注释 |
| 选择 | |
| Alt+Shift+↑ | 选择封装元素 |
| Alt+Shift+← | 选择上一个元素 |
| Alt+Shift+→ | 选择下一个元素 |
| Shift+← | 从光标处开始往左选择字符 |
| Shift+→ | 从光标处开始往右选择字符 |
| Ctrl+Shift+← | 选中光标左边的单词 |
| Ctrl+Shift+→ | 选中光标又边的单词 |
| 移动 | |
| Ctrl+← | 光标移到左边单词的开头，相当于vim的b |
| Ctrl+→ | 光标移到右边单词的末尾，相当于vim的e |
| 搜索 | |
| Ctrl+K | 参照选中的Word快速定位到下一个（如果没有选中word，则搜索上一次使用搜索的word） |
| Ctrl+Shift+K | 参照选中的Word快速定位到上一个 |
| Ctrl+J | 正向增量查找（按下Ctrl+J后,你所输入的每个字母编辑器都提供快速匹配定位到某个单词,如果没有,则在状态栏中显示没有找到了,查一个单词时,特别实用,要退出这个模式，按escape建） |
| Ctrl+Shift+J | 反向增量查找（和上条相同,只不过是从后往前查） |
| Ctrl+Shift+U | 列出所有包含字符串的行 |
| Ctrl+H | 打开搜索对话框 |
| Ctrl+G | 工作区中的声明 |
| Ctrl+Shift+G | 工作区中的引用 |
| 导航 | |
| Ctrl+Shift+T | 搜索类（包括工程和关联的第三jar包） |
| Ctrl+Shift+R | 搜索工程中的文件 |
| Ctrl+E | 快速显示当前Editer的下拉列表（如果当前页面没有显示的用黑体表示） |
| F4 | 打开类型层次结构 |
| | |

| | |
|---|---|
| F3 | 跳转到声明处 |
| Alt+← | 前一个编辑的页面 |
| Alt+→ | 下一个编辑的页面（当然是针对上面那条来说了） |
| Ctrl+PageUp/PageDown | 在编辑器中，切换已经打开的文件 |
| 调试 | |
| F5 | 单步跳入 |
| F6 | 单步跳过 |
| F7 | 单步返回 |
| F8 | 继续 |
| Ctrl+Shift+D | 显示变量的值 |
| Ctrl+Shift+B | 在当前行设置或者去掉断点 |
| Ctrl+R | 运行至行(超好用，可以节省好多的断点) |
| 重构（一般重构的快捷键都是Alt+Shift开头的了） | |
| Alt+Shift+R | 重命名方法名、属性或者变量名 （是我自己最爱用的一个了,尤其是变量和类的Rename,比手工方法能节省很多劳动力） |
| Alt+Shift+M | 把一段函数内的代码抽取成方法 （这是重构里面最常用的方法之一了,尤其是对一大堆泥团代码有用） |
| Alt+Shift+C | 修改函数结构（比较实用,有N个函数调用了这个方法,修改一次搞定） |
| Alt+Shift+L | 抽取本地变量（ 可以直接把一些魔法数字和字符串抽取成一个变量,尤其是多处调用的时候） |
| Alt+Shift+F | 把Class中的local变量变为field变量 （比较实用的功能） |
| Alt+Shift+I | 合并变量（可能这样说有点不妥Inline） |
| Alt+Shift+V | 移动函数和变量（不怎么常用） |
| Alt+Shift+Z | 重构的后悔药（Undo） |
| 其他 | |
| Alt+Enter | 显示当前选择资源的属性，windows下的查看文件的属性就是这个快捷键，通常用来查看文件在windows中的实际路径 |
| Ctrl+↑ | 文本编辑器 上滚行 |

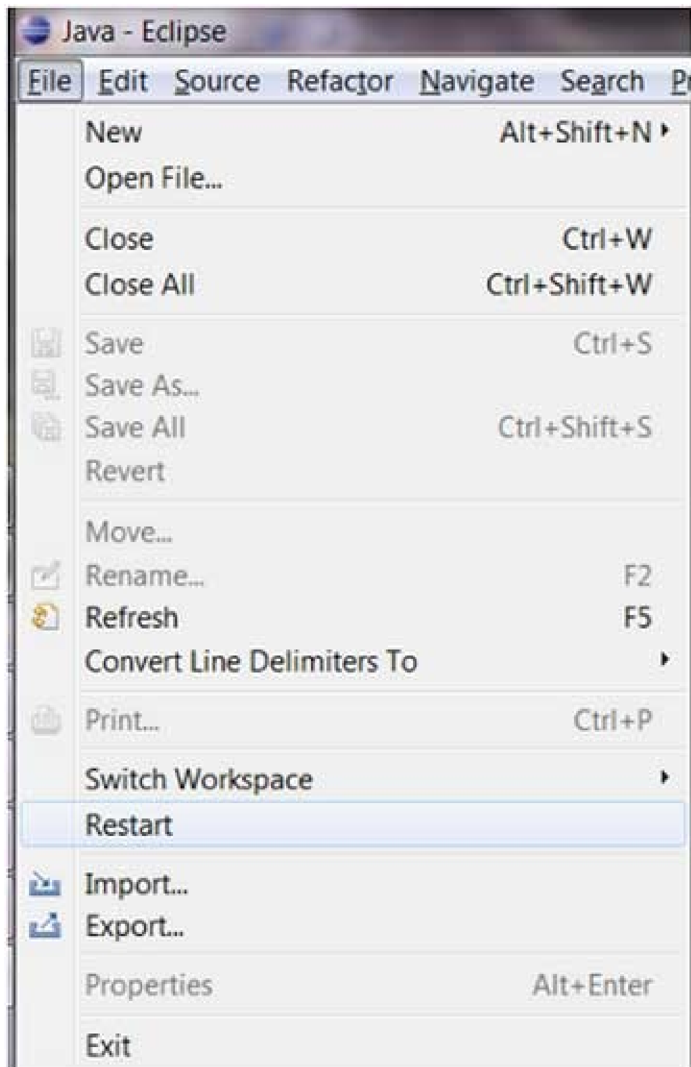
| | |
|---|---|
| Ctrl+↓ | 文本编辑器 下滚行 |
| Ctrl+M | 最大化当前的Edit或View （再按则反之） |
| Ctrl+O | 快速显示 OutLine（不开Outline窗口的同学，这个快捷键是必不可少的） |
| Ctrl+T | 快速显示当前类的继承结构 |
| Ctrl+W | 关闭当前Editer（windows下关闭打开的对话框也是这个，还有qq、旺旺、浏览器等都是） |
| Ctrl+L | 文本编辑器 转至行 |
| F2 | 显示工具提示描述 |

# Eclipse 重启选项

## 重启 Eclipse

重启选项允许用户重启 Eclipse。

我们可以通过点击 File 菜单选择 Restart 菜单项来重启 Eclipse。

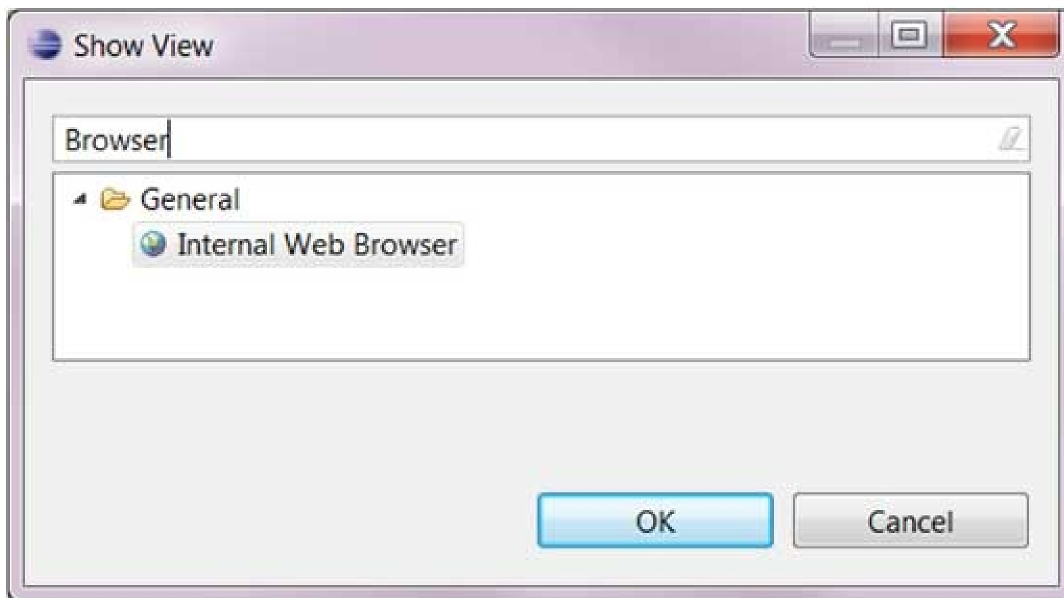



在安装插件后，用户一般都会被提醒要重启 Eclipse。如果用户当时没有重启 Eclipse，可以通过该选项来重启。

# Eclipse 内置浏览器

## Web 浏览器

Eclipse 系统内部自带了浏览器，该浏览器可以通过点击 Window 菜单并选择 Show View > Other，在弹出来的对话框的搜索栏中输入 "browser"。




在树形菜单中选择 "Internal Web Browser" 并点击 OK。

在内置浏览器中我们在地址栏中输入网址，如：http://www.w3cschool.cc，即可打开网页。



# EJB教程 - EJB

企业Java Beans（EJB）是一个集开发构建高度可扩展性和强大的企业级应用程序的架构上部署符合J2EE规范的应用服务器，如JBOSS，网站逻辑等。 EJB3.0从EJB2.0是一个伟大的转变，使得基于EJB应用的开发变得更容易。本教程讲述EJB概念和理解，在这一个过程中需要创建和部署企业级应用程序启动和运行。 读者对象： 本教程是专为软件专业人员学习EJB编程简单入门。本教程将介绍EJB编程概念和理解，在完成本教程后，希望你能够对EJB有一个初级的认识和应用。

前置技术知识要求： 在继续本教程之前，您应该对Java编程语言有一定的了解，文本编辑器和执行程序这些也必不可少，因为我们要开发企业应用程序使用EJB，如果你已经了解其他技术，如数据库服务器，应用服务器，那么你可以跳过上面技术知识的学习，直接进入下一章节。

# EJB概述 - EJB

EJB其实就是企业Java Beans。 EJB是J2EE平台的重要组成部分。 J2EE平台基于组件的企业级应用架构，提供多层次，分布式和高事务的功能特点。 EJB提供了一个架构，充分考虑健壮性，高可扩展性和高性能的基于组件的企业应用程序开发和部署。一个EJB应用程序可以部署在任何符合J2EE1.3标准规范的应用程序服务器上。我们将在本教程中讨论**EJB3**.0。

优点

- 简化开发大型企业级应用。
- 应用服务器/ EJB容器提供了系统级服务，如事务处理，日志，负载均衡，持久性机制，异常处理等。开发者只专注于业务逻辑的应用程序。
- EJB容器管理EJB实例的生命周期，因此，开发人员并不需要担心何时创建/删除EJB对象。

类型 EJB主要有三种类型，下面简要介绍：

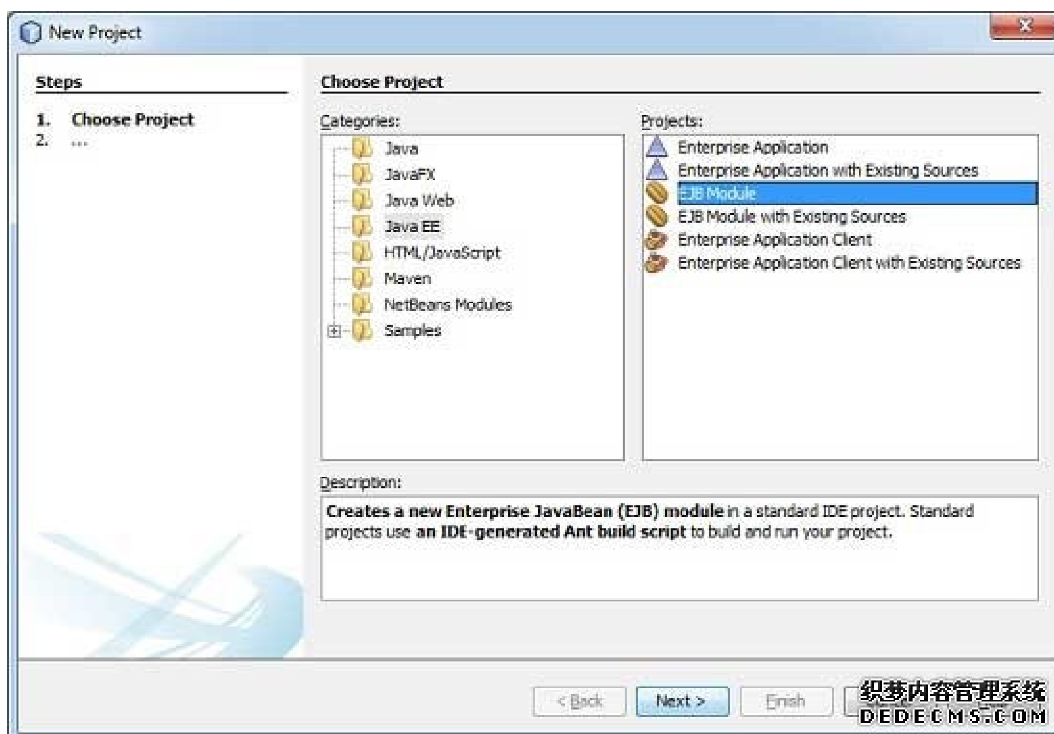
| 类型 | 描述 |
| --- | --- |
| Session Bean | 会话bean将特定用户的数据存储为一个单一的会话。它可以是状态或无状态。它占用更少资源，相比于实体Bean。只要终止用户会话，会话bean被销毁。 |
| Entity Bean | 实体bean代表持久性数据存储。可将用户数据保存到数据库中，通过实体bean，后来就可以检索从数据库中的实体bean。 |
| Message Driven Bean | 使用消息驱动bean上下文中的JMS（Java消息服务）。消息驱动Bean可以从外部实体消耗JMS消息，并采取相应的动作。 |

# EJB创建应用 - EJB

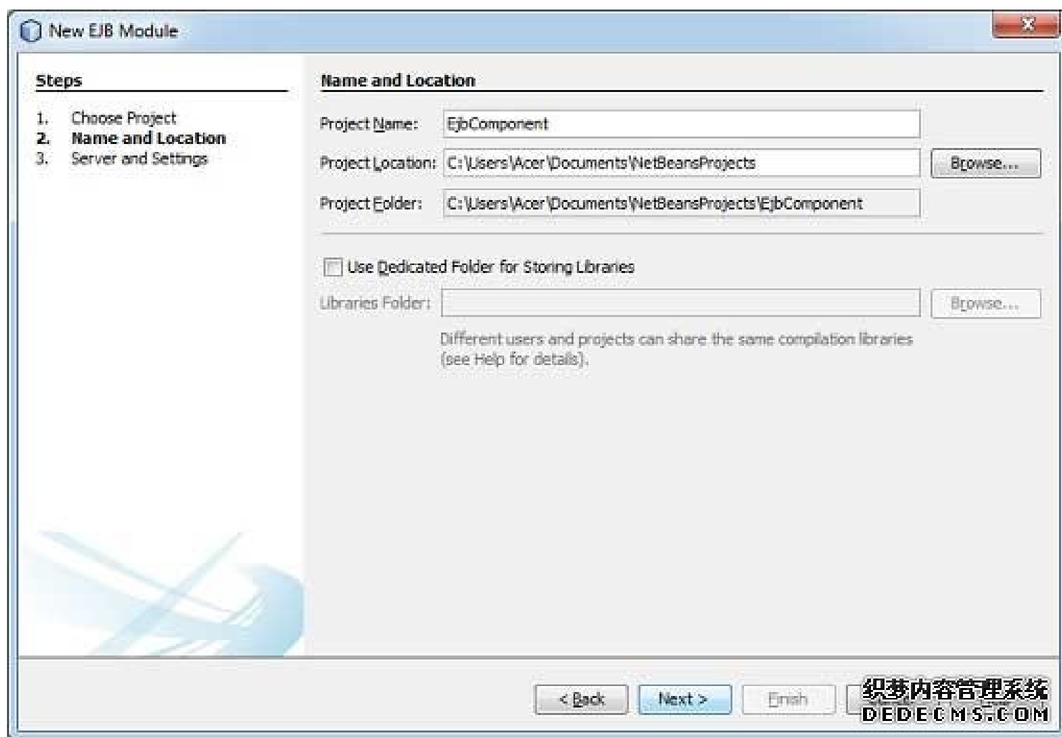
要创建一个简单的EJB模块，我们这里使用NetBeans“New project”向导。在下面的例子中，我们将创建一个名为“Component”的EJB模块项目。

## 创建项目

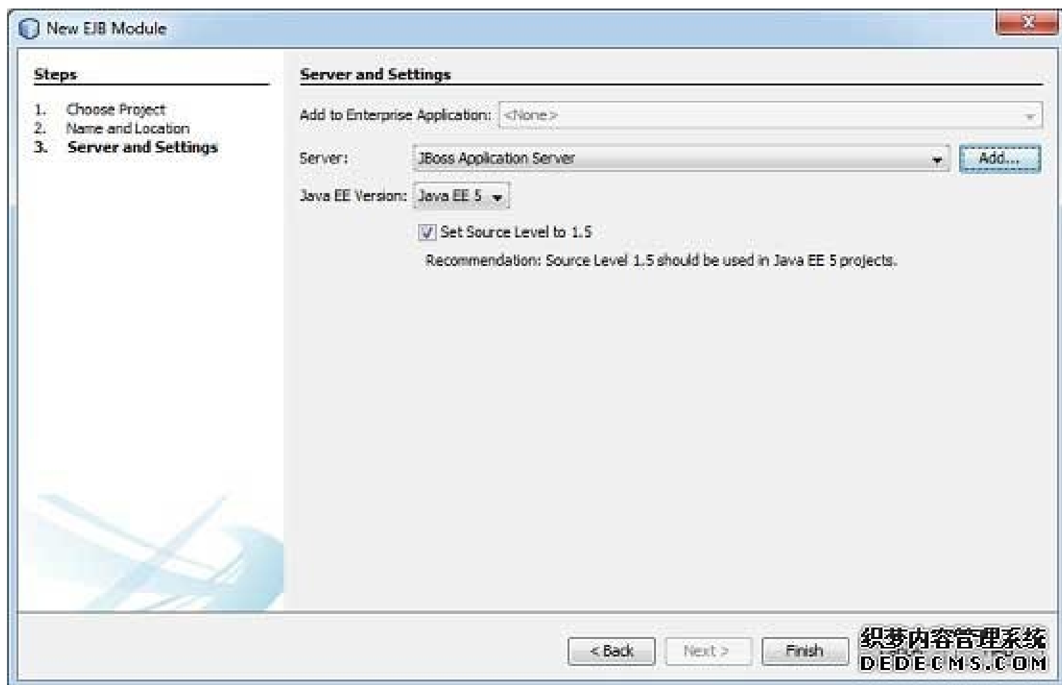
在NetBeans IDE中，选择**File > New Project >**. 可以看到如下图.




在类别中选择项目类型，Java EE的EJB模块项目类型。点击Next>按钮，你会看到以下的画面。




输入项目的名称和位置。点击Next>按钮。你会看到以下的画面。



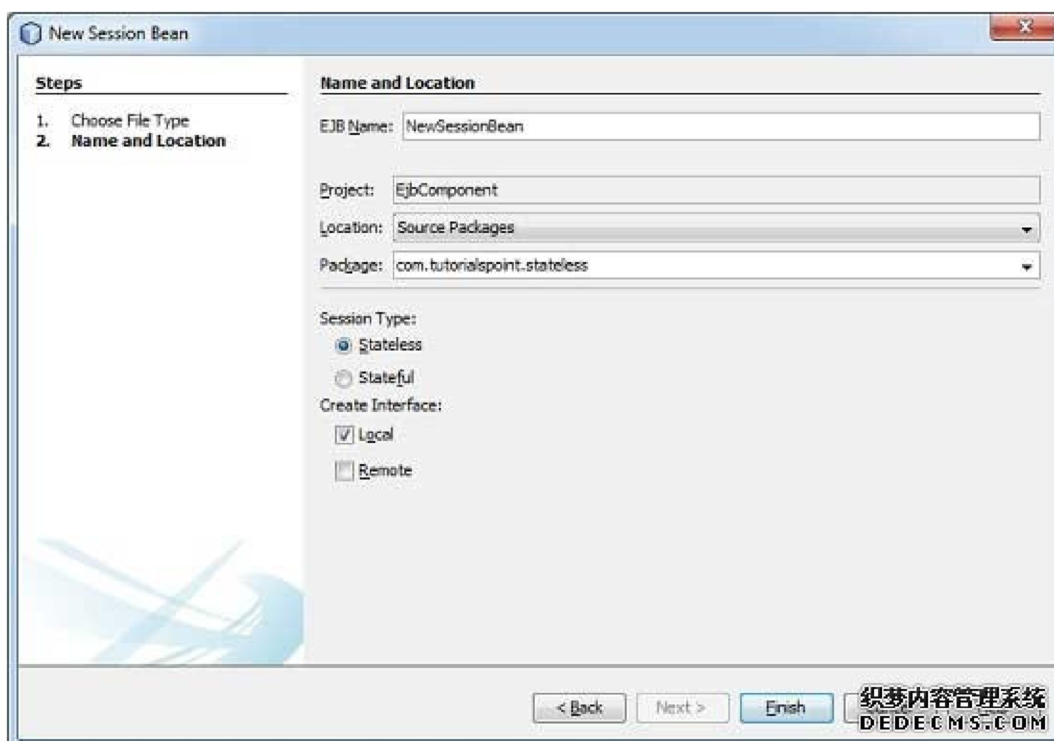

选择服务器为JBoss应用服务器。单击“**Finish**”按钮。你会看到以下由NetBeans创建的项目。




# 创建一个简单的EJB

要创建一个简单的EJB，我们将使用NetBeans“**New**”向导。在下面的例子中，我们将在EjbComponent项目下创建一个无状态EJB类名为**librarySessionBean**。

在项目资源管理器窗口中选择项目EjbComponent，右键单击它。选择**New > Session Bean**。您将看到新的会话Bean向导。




输入会话bean的名称和包名。单击“**Finish** ”按钮。你会看到以下由NetBeans创建的EJB类。

- **LibrarySessionBean** - 无状态会话bean
- **LibrarySessionBeanLocal** - 本地接口的会话bean

要改变本地接口，我们要一个基于控制台的应用程序访问我们的EJB远程接口。远程/本地接口用于公开一个EJB的业务方法实现。

LibrarySessionBeanLocal更名为LibrarySessionBeanRemote和LibrarySessionBean实现LibrarySessionBeanRemote接口。

*LibrarySessionBeanRemote*

```
package com.tutorialspoint.stateless;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibrarySessionBeanRemote {

    void addBook(String bookName);

    List getBooks();

}
```

*LibrarySessionBean*

```
package com.tutorialspoint.stateless;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;

@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote

    List<String> bookShelf;

    public LibrarySessionBean(){
       bookShelf = new ArrayList<String>();
    }

    public void addBook(String bookName) {
        bookShelf.add(bookName);
    }

    public List<String> getBooks() {
        return bookShelf;
    }
}
```

## 构建项目

- 在**Project Explorer**窗口中选择EjbComponent项目。
- 右键点击它来打开上下文菜单。
- 选择 **clean and build**。

在NetBeans控制台输出，你会看到以下的输出。

```
ant -f C:\EJB\EjbComponent clean dist
init:
undeploy-clean:
deps-clean:
Deleting directory C:EJBEjbComponentuild
Deleting directory C:EJBEjbComponentdist
clean:
init:
deps-jar:
Created dir: C:EJBEjbComponentuildclasses
Copying 3 files to C:EJBEjbComponentuildclassesMETA-INF
Created dir: C:EJBEjbComponentuildempty
Created dir: C:EJBEjbComponentuildgenerated-sourcesap-source-output
Compiling 2 source files to C:EJBEjbComponentuildclasses
warning: [options] bootstrap class path not set in conjunction with
Note: C:EJBEjbComponentsrcjavacom    utorialspointstateless
LibraryPersistentBean.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 warning
compile:
library-inclusion-in-archive:
Created dir: C:EJBEjbComponentdist
Building jar: C:EJBEjbComponentdistEjbComponent.jar
dist:
BUILD SUCCESSFUL (total time: 3 seconds)
```

## 启动应用程序服务器

- 服务器在服务窗口下选择JBoss应用服务器。
- 右键点击它来打开上下文菜单。
- 选择**start**。

你会看到下面的输出在NetBeans中JBoss应用服务器下的输出。

```
Calling C:jboss-5.1.0.GAin
un.conf.bat
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: C:jboss-5.1.0.GA

  JAVA: C:Program Files (x86)Javajdk1.6.0_21injava

  JAVA_OPTS: -Dprogram.name=run.bat -Xms128m -Xmx512m -server

  CLASSPATH: C:jboss-5.1.0.GAin
un.jar

=========================================================================

16:25:50,062 INFO  [ServerImpl] Starting JBoss (Microcontainer)...
16:25:50,062 INFO  [ServerImpl] Release ID: JBoss [The Oracle] 5.1.
...

16:26:40,420 INFO  [TomcatDeployment] deploy, ctxPath=/admin-consol
16:26:40,485 INFO  [config] Initializing Mojarra (1.2_12-b01-FCS) f
16:26:42,362 INFO  [TomcatDeployment] deploy, ctxPath=/
16:26:42,406 INFO  [TomcatDeployment] deploy, ctxPath=/jmx-console
16:26:42,471 INFO  [Http11Protocol] Starting Coyote HTTP/1.1 on htt
16:26:42,487 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127
16:26:42,493 INFO  [ServerImpl] JBoss (Microcontainer) [5.1.0.GA (b
```

# 部署项目

- 在**Project Explorer**窗口中选择EjbComponent项目。
- 右击它，打开上下文菜单。
- 选择 **Deploy**.

在NetBeans控制台输出，你会看到下面的输出。

```
ant -f C:\EJB\EjbComponent -DforceRedeploy=true -Ddirectory.deploym
init:
deps-jar:
compile:
library-inclusion-in-archive:
Building jar: C:EJBEjbComponentdistEjbComponent.jar
dist-directory-deploy:
pre-run-deploy:
Checking data source definitions for missing JDBC drivers...
Distributing C:EJBEjbComponentdistEjbComponent.jar to [org.jboss.de
Deploying C:EJBEjbComponentdistEjbComponent.jar
Applicaton Deployed
Operation start started
Operation start completed
post-run-deploy:
run-deploy:
run:
BUILD SUCCESSFUL (total time: 2 seconds)
```

JBoss应用服务器的日志输出

```
16:30:00,963 INFO  [DeployHandler] Begin start, [EjbComponent.jar]
...
16:30:01,233 INFO  [Ejb3DependenciesDeployer] Encountered deploymen
...
16:30:01,281 INFO  [JBossASKernel]    jndi:LibrarySessionBean/remot
16:30:01,281 INFO  [JBossASKernel]    Class:com.tutorialspoint.stat
16:30:01,281 INFO  [JBossASKernel]    jndi:LibrarySessionBean/remot
16:30:01,281 INFO  [JBossASKernel]  Added bean(jboss.j2ee:jar=EjbCo
LibrarySessionBean,service=EJB3) to KernelDeployment of: EjbCompone
16:30:01,282 INFO  [JBossASKernel] installing bean: jboss.j2ee:jar=
16:30:01,282 INFO  [JBossASKernel]   with dependencies:
16:30:01,282 INFO  [JBossASKernel]   and demands:
16:30:01,282 INFO  [JBossASKernel]    jboss.ejb:service=EJBTimerSer
...
16:30:01,283 INFO  [EJB3EndpointDeployer] Deploy AbstractBeanMetaDa
...
16:30:01,394 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:01,395 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibrarySessionBean/remote - EJB3.x Default Remote Business Inter
   LibrarySessionBean/remote-com.tutorialspoint.stateless.LibrarySe
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibrarySessionBean/remote - EJB3.x Default Remote Business Inter
   LibrarySessionBean/remote-com.tutorialspoint.stateless.LibrarySe
```

# 创建客户端访问EJB

- 在NetBeans IDE中选择 **File > New Project >**.
- 类别下选择项目类型为**Java**，项目类型为Java应用程序的Java。点击**Next>**按钮。
- 输入项目的名称和位置。单击“**Finish >**“按钮。我们选择名为EjbTester。
- 右键点击项目名称(在Project explore窗口中)。选择属性**properties**。
- 添加EJB组件项目的库使用“**Add Project** ”按钮，在**compile**选项卡下创建的。
- 添加JBoss库使用添加**Add jar/folder**按钮，在**compile**选项卡。 Jboss的库可以位于<JBOSS安装文件夹>客户端文件夹。

在工程中创建 jndi.properties说一个句话 EjbTester.

*jndi.properties*

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

创建包com.tutorialspoint.test和EJBTester.java类在下面。

*EJBTester.java*

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateless.LibrarySessionBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
      props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
```

```
         ex.printStackTrace();
         }
         try {
         ctx = new InitialContext(props);
         } catch (NamingException ex) {
         ex.printStackTrace();
         }
         brConsoleReader =
         new BufferedReader(new InputStreamReader(System.in));
      }
      public static void main(String[] args) {

         EJBTester ejbTester = new EJBTester();

         ejbTester.testStatelessEjb();
      }
      private void showGUI(){
         System.out.println("**********************");
         System.out.println("Welcome to Book Store");
         System.out.println("**********************");
         System.out.print("Options
   1\. Add Book
   2\. Exit
   Enter Choice: ");
      }
      private void testStatelessEjb(){
         try {
            int choice = 1;
            LibrarySessionBeanRemote libraryBean =
            (LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBean/r
            while (choice != 2) {
               String bookName;
               showGUI();
               String strChoice = brConsoleReader.readLine();
               choice = Integer.parseInt(strChoice);
               if (choice == 1) {
                  System.out.print("Enter book name: ");
                  bookName = brConsoleReader.readLine();
                  libraryBean.addBook(bookName);
               }else if (choice == 2) {
                  break;
               }
            }
            List<String> booksList = libraryBean.getBooks();
            System.out.println("Book(s) entered so far: " + booksList.
            for (int i = 0; i < booksList.size(); ++i) {
            System.out.println((i+1)+". " + booksList.get(i));
            }
            LibrarySessionBeanRemote libraryBean1 =
            (LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBean/r
            List<String> booksList1 = libraryBean1.getBooks();
            System.out.println(
            "***Using second lookup to get library stateless object***
```

```
            System.out.println(
            "Book(s) entered so far: " + booksList1.size());
            for (int i = 0; i < booksList1.size(); ++i) {
               System.out.println((i+1)+". " + booksList1.get(i));
            }
         } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }finally {
            try {
               if(brConsoleReader !=null){
                  brConsoleReader.close();
               }
            } catch (IOException ex) {
               System.out.println(ex.getMessage());
            }
         }
      }
   }
```

## 运行客户端访问EJB

在**project explorer**中找到EJBTester.java。右键点击上EJBTester类，并选择“**run file**”。

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. Learn Java
***Using second lookup to get library stateless object***
Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 13 seconds)
```

在下面的章节中，我们将讨论这个完整的EJB的多个层面应用程序。

# EJB无状态Bean - EJB

无状态会话bean是一种企业bean，通常是用于执行独立操作。正如它的名字一样，无状态会话bean不具有任何关联的客户端的状态，但它可能会保持其实例的状态。 EJB容器通常创建一个容器池和几无状态的bean的对象，并使用这些对象来处理客户端的请求。由于有容器池，实例变量的值不能保证跨查找/方法调用同一个。

下面是创建一个无状态的EJB所需的步骤。

- 创建一个远程/本地接口暴露的业务方法。
- 此接口将用于EJB客户端应用程序。
- 使用@ Local注释如果EJB客户端是在相同的环境中部署EJB会话Bean。
- 使用@ Remote批注如果EJB客户端是在不同的环境中部署EJB会话Bean。
- 创建一个无状态会话bean实现上述接口。
- 使用@ Stateless注释，以表示它一个无状态的bean。 EJB容器会自动创建通过读取这个注解，在部署过程中的相关配置或接口。

*Remote Interface*

```
import javax.ejb.Remote;

@Remote
public interface LibrarySessionBeanRemote {
   //add business method declarations
}
```

*Stateless EJB*

```
@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote
   //implement business method
}
```

## 示例应用程序

让我们创建一个测试测试无状态EJB的EJB应用程序。

| 步骤 | 描述 |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.stateless* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand stateless ejb concepts. |
| 2 | Create *LibrarySessionBean.java* and *LibrarySessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged. |
| 3 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 4 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 5 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

## EJBComponent (EJB Module)

## LibrarySessionBeanRemote.java

```
package com.tutorialspoint.stateless;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibrarySessionBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

## LibrarySessionBean.java

```
package com.tutorialspoint.stateless;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;

@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote

   List<String> bookShelf;

   public LibrarySessionBean(){
      bookShelf = new ArrayList<String>();
   }

   public void addBook(String bookName) {
      bookShelf.add(bookName);
   }

   public List<String> getBooks() {
      return bookShelf;
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibrarySessionBean/remote**.
- We'll using this lookup string to get remote business object of type - **com.tutorialspoint.stateless.LibrarySessionBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibrarySessionBean/remote - EJB3.x Default Remote Business Inter
   LibrarySessionBean/remote-com.tutorialspoint.stateless.LibrarySe
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibrarySessionBean/remote - EJB3.x Default Remote Business Inter
   LibrarySessionBean/remote-com.tutorialspoint.stateless.LibrarySe
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service
- InitialContext object will be used to lookup stateless session bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibrarySessionBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
```

```
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testStatelessEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testStatelessEjb(){

      try {
         int choice = 1;

         LibrarySessionBeanRemote libraryBean =
         LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBean/re

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
```

```
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
         LibrarySessionBeanRemote libraryBean1 =
            (LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBea
         List<String> booksList1 = libraryBean1.getBooks();
         System.out.println(
            "***Using second lookup to get library stateless object
         System.out.println(
            "Book(s) entered so far: " + booksList1.size());
         for (int i = 0; i < booksList1.size(); ++i) {
            System.out.println((i+1)+". " + booksList1.get(i));
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester在执行以下任务。

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatelessEjb() method, jndi lookup is done with name - "LibrarySessionBean/remote" to obtain the remote business object (stateless ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.

- If user enters 1, system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in its instance variable.
- If user enters 2, system retrieves books using stateless session bean getBooks() method and exits.
- Then another jndi lookup is done with name - "LibrarySessionBean/remote" to obtain the remote business object (stateless ejb) again and listing of books is done.

## 运行客户端访问EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. Learn Java
***Using second lookup to get library stateless object***
Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 13 seconds)
```

## 再次运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 0
***Using second lookup to get library stateless object***
Book(s) entered so far: 1
1\. Learn Java
BUILD SUCCESSFUL (total time: 12 seconds)
```

- 如上图所示的输出可能会有所不同，这取决于许多无状态JBoss的EJB对象保持。
- 万一单一的无状态EJB对象得以维持，每次查找后，你可能会看到相同书籍列表。
- EJB容器可以为每个查询返回相同的无状态EJB对象。

无状态EJB的bean实例变量的值是直到重新启动服务器才失效的。

# EJB有状态Bean - EJB

有状态会话Bean是一种企业bean保存客户端的会话状态类型。有状态会话bean作为每它的名字相关的客户端状态保持在它的实例变量。 EJB容器创建一个单独的有状态会话bean来处理客户端的每个请求。只要请求范围过，有状态会话bean被销毁。

以下是创建一个有状态的EJB所需的步骤：

- Create a remote/local interface exposing the business methods.
- This interface will be used by the ejb client application.
- Use @Local annotation if ejb client is in same environment where ejb session bean is to be deployed.
- Use @Remote annotation if ejb client is in different environment where ejb session bean is to be deployed.
- Create a stateful session bean implementing the above interface.
- Use @Stateful annotation to signify it a stateful bean. EJB Container automatically creates the relevant configurations or interfaces required by reading this annotation during deployment.

远程接口

```
import javax.ejb.Remote;

@Remote
public interface LibraryStatefulSessionBeanRemote {
   //add business method declarations
}
```

*Stateful EJB*

```
@Stateful
public class LibraryStatefulSessionBean implements LibraryStatefulS
   //implement business method
}
```

## 示例应用程序

让我们创建一个测试测试状态EJB的EJB应用程序。

| Step | 描述 |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.stateful* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand stateful ejb concepts. |
| 2 | Create *LibraryStatefulSessionBean.java* and *LibraryStatefulSessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged. |
| 3 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 4 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 5 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

# EJBComponent (EJB Module)

# LibraryStatefulSessionBeanRemote.java

```
package com.tutorialspoint.stateful;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryStatefulSessionBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

# LibraryStatefulSessionBean.java

```
package com.tutorialspoint.stateful;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateful;

@Stateful
public class LibraryStatefulSessionBean implements LibraryStatefulS

   List<String> bookShelf;

   public LibraryStatefulSessionBean(){
      bookShelf = new ArrayList<String>();
   }

   public void addBook(String bookName) {
      bookShelf.add(bookName);
   }

   public List<String> getBooks() {
      return bookShelf;
   }
}
```

- 只要你部署在JBoss EjbComponent项目发现jboss的日志。
- JBoss已经自动创建一个JNDI条目会话bean-LibraryStatefulSessionBean/remote。
- 我们将使用这个查询字符串来获得远程类型的业务对象-**com.tutorialspoint.stateful.LibraryStatefulSessionBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibraryStatefulSessionBean/remote - EJB3.x Default Remote Busine
   LibraryStatefulSessionBean/remote-com.tutorialspoint.stateful.Li
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibraryStatefulSessionBean/remote - EJB3.x Default Remote Busine
   LibraryStatefulSessionBean/remote-com.tutorialspoint.stateful.Li
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service
- InitialContext object will be used to lookup stateful session bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryStatefulSessionBeanRemote
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
```

```
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testStatelessEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testStatelessEjb(){

      try {
         int choice = 1;

         LibraryStatefulSessionBeanRemote libraryBean =
         LibraryStatefulSessionBeanRemote)ctx.lookup("LibraryStatef

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
         LibraryStatefulSessionBeanRemote libraryBean1 =
            (LibraryStatefulSessionBeanRemote)ctx.lookup("LibrarySt
```

```
            List<String> booksList1 = libraryBean1.getBooks();
            System.out.println(
               "***Using second lookup to get library stateful object
            System.out.println(
               "Book(s) entered so far: " + booksList1.size());
            for (int i = 0; i < booksList1.size(); ++i) {
               System.out.println((i+1)+". " + booksList1.get(i));
            }
         } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }finally {
            try {
               if(brConsoleReader !=null){
                  brConsoleReader.close();
               }
            } catch (IOException ex) {
               System.out.println(ex.getMessage());
            }
         }
      }
   }
```

EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and saves the book using stateful session bean addBook() method. Session Bean is storing the book in its instance variable.
- If user enters 2, system retrieves books using stateful session bean getBooks() method and exits.
- Then another jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful ejb) again and listing of books is done.

# 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键单击类EJBTester并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. Learn Java
***Using second lookup to get library stateful object***
Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 13 seconds)
```

## 再次运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键单击类EJBTester选择**run file**.

在NetBeans控制台验证以下输出.

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 0
***Using second lookup to get library stateful object***
Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 12 seconds)
```

- 输出上面显示的状态，对于每个查询不同状态的EJB实例将被返回.
- 只适用于单个会话状态EJB对象值。在第二次运行时，我们没有得到任何值.

# EJB持久性 - EJB

EJB2.0中使用的实体bean持久化机制在很大程度上被EJB3.0取代。现在实体bean是一个简单的POJO映射表。

以下是持久性API的关键角色

- **Entity** - 持久对象代表数据存储记录。这也是可序列化的。
- **EntityManager** - 持久性接口做数据操作，如添加/删除/更新/找到持久化对象（实体）。它还有助于执行查询使用query接口
- **Persistence unit (persistence.xml)** - 持久性单元介绍了持久性机制的属性。
- **Data Source (*ds.xml)** - 数据源描述了数据存储相关的属性，如连接URL。用户名，密码等。

为了证明EJB的持久化机制，我们要做好以下几项工作。

- Step 1. 在数据库中创建表.
- Step 2. 创建实体类对应的表.
- Step 3. 创建数据源和持久性单元
- Step 4. 创建一个无状态EJB EntityManager实例.
- Step 5. 更新无状态EJB。添加添加记录并获得通过实体管理器从数据库中记录的方法。
- Step 6. 一个基于控制台应用程序客户端将访问无状态EJB的持久化数据库中的数据。

## 创建表

创建一个表 **books** 在默认的数据库 **postgres**.

```
CREATE TABLE books (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
```

## 创建实体类

```
//mark it entity using Entity annotation
//map table name using Table annoation
@Entity
@Table(name="books")
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   //mark id as primary key with autogenerated value
   //map database column id with id field
   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }
   ...
}
```

## 创建数据源和持久性单元

DataSource (jboss-ds.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
   <local-tx-datasource>
      <jndi-name>PostgresDS</jndi-name>
      <connection-url>jdbc:postgresql://localhost:5432/postgres</co
      <driver-class>org.postgresql.driver</driver-class>
      <user-name>sa</user-name>
      <password>sa</password>
      <min-pool-size>5</min-pool-size>
      <max-pool-size>20</max-pool-size>
      <idle-timeout-minutes>5</idle-timeout-minutes>
   </local-tx-datasource>
</datasources>
```

Persistence Unit (persistence.xml)

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persis
   <persistence-unit name="EjbComponentPU" transaction-type="JTA">
      <jta-data-source>java:/PostgresDS</jta-data-source>
      <exclude-unlisted-classes>false</exclude-unlisted-classes>
      <properties/>
   </persistence-unit>
   <persistence-unit name="EjbComponentPU2" transaction-type="JTA">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:/PostgresDS</jta-data-source>
      <exclude-unlisted-classes>false</exclude-unlisted-classes>
      <properties>
         <property name="hibernate.hbm2ddl.auto" value="update"/>
      </properties>
   </persistence-unit>
</persistence>
```

## Create Stateless EJB having EntityManager instance

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBean

   //pass persistence unit to entityManager.
   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Books").getResultList(
   }
   ...
}
```

构建EJB模块后，我们需要一个无状态的bean，我们将在下一节要创建客户端来访问。

## 示例应用程序

让我们创建一个测试EJB应用程序来测试EJB的持久化机制。

| Step | 描述 |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand ejb persistence concepts. |
| 2 | Create *Book.java* under package *com.tutorialspoint.entity* and modify it as shown below. |
| 3 | Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as explained in the *EJB - Create Application* chapter and modify them as shown below. |
| 4 | Create *jboss-ds.xml* in **EjbComponent > setup** folder and *persistence.xml* in **EjbComponent > src > conf** folder. These folder can be seen in files tab in Netbeans. Modify these files as shown above. |
| 5 | 清理并生成应用程序以确保业务逻辑是按要求工作。 |
| 6 | 最后，将应用程序部署在JBoss应用服务器上的jar文件的形式。 JBoss应用服务器将自动开始浏览网页，如果它尚未启动。 |
| 7 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below. |

# EJBComponent (EJB Module)

# Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="books")
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

## LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

# LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBean

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList(
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.

- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We'll using this lookup string to get remote business object of type - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the followin
   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.stateless.Librai
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the followin

   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.stateless.Librai
...
```

## EJBTester (EJB Client)

## jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service
- InitialContext object will be used to lookup stateless session bean

## EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEntityEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testEntityEjb(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
         LibraryPersistentBeanRemote)ctx.lookup("LibraryPersistentE

         while (choice != 2) {
```

```
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.

- If user enters 1, system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is persisting the book in database via EntityManager call.
- If user enters 2, system retrieves books using stateful session bean getBooks() method and exits.
- Then another jndi lookup is done with name - "LibraryStatelessSessionBean/remote" to obtain the remote business object (stateless ejb) again and listing of books is done.

## 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择 **run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. learn java
BUILD SUCCESSFUL (total time: 15 seconds)
```

## 再次运行客户端来访问EJB。

访问EJB之前重新启动JBoss。

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择 **run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Spring
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 2
1\. learn java
2\. Learn Spring
BUILD SUCCESSFUL (total time: 15 seconds)
```

- 上面显示的输出状态书存储在持久性存储，并从数据库中检索。

# EJB消息驱动Bean - EJB

一个消息驱动bean是一种类型的企业Bean，这是由EJB容器调用，当它接收到一个消息队列或主题。消息驱动bean是一个无状态的bean是用来做异步任务。

为了演示如何使用消息驱动bean，我们将利用EJB持久章节的内容，我们要做好以下几项工作。

- Step 1. 在数据库中创建表(请参阅EJB持久章节)。
- Step 2. 创建实体类对应的表 (请参阅EJB持久章节).
- Step 3. 创建数据源和持久性单元 (请参阅EJB持久章节).
- Step 4. 创建一个无状态EJB EntityManager实例 (请参阅EJB持久章节).
- Step 5. 更新无状态EJB。添加添加记录的方法，并通过实体管理器，从数据库中获取记录 (请参阅EJB持久章节).
- Step 6. 创建队列名为 **BookQueue** 在JBoss **default** 应用目录.
- Step 7. 一个基于控制台应用程序客户端发送消息到这个队列
- Step 8. 创建消息驱动bean将使用无状态的bean持久化客户数据。
- Step 9. JBoss的EJB容器将调用上面的消息驱动bean，并把它传递的消息将要发送给客户端。

## 创建队列

创建一个文件名为jbossmq目的地service.xml中，如果不存在 **<JBoss Installation Folder> > server > default > deploy** 目录.

在这里，我们创建名为BookQueue一个队列

jbossmq-destinations-service.xml

```
<mbean code="org.jboss.mq.server.jmx.Queue"
   name="jboss.mq.destination:service=Queue,name=BookQueue">
   <depends optional-attribute-name="DestinationManager">
      jboss.mq:service=DestinationManager
   </depends>
</mbean>
```

当你启动JBoss，你会看到类似的内容在JBoss日志

```
...
10:37:06,167 INFO  [QueueService] Queue[/queue/BookQueue] started,
...
```

## 创建消息驱动bean

```
@MessageDriven(
   name = "BookMessageHandler",
   activationConfig = {
      @ActivationConfigProperty( propertyName = "destinationType",
                                 propertyValue = "javax.jms.Queue")
      @ActivationConfigProperty( propertyName = "destination",
                                 propertyValue ="/queue/BookQueue")
   }
)
public class LibraryMessageBean implements MessageListener {

   @Resource
   private MessageDrivenContext mdctx;

   @EJB
   LibraryPersistentBeanRemote libraryBean;

   public LibraryMessageBean(){
   }

   public void onMessage(Message message) {
   }
}
```

- LibraryMessageBean annoatated@MessageDriven注解，把它标记为消息驱动bean。
- Its properties are defined as destinationType - Queue and destination - /queue/BookQueue.
- It implements MessageListener interface which exposes onMessage method.
- It has MessgeDrivenContext as resource.
- LibraryPersistentBeanRemote stateless bean is injected in this bean for persistence purpose.

构建EjbComponent项目，并将其部署在JBoss上。构建和部署EJB模块后，我们需要一个客户端发送一个消息到JBoss队列。

# 示例应用程序

让我们创建一个测试EJB应用程序来测试消息驱动bean。

| Step | Description |
| --- | --- |
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand ejb persistence concepts. |
| 2 | Create *Book.java* under package *com.tutorialspoint.entity* as created in _EJB-Persistence_chapter |
| 3 | Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as created in *EJB-Persistence* chapter |
| 4 | Create *jboss-ds.xml* in **EjbComponent > setup** folder and *persistence.xml* in **EjbComponent > src > conf** folder. These folder can be seen in files tab in Netbeans as created in *EJB-Persistence* chapter |
| 5 | Create *LibraryMessageBean.java* under a package *com.tutorialspoint.messagebean* and modify it as shown below. |
| 6 | Create *BookQueue* queue in Jboss as described above. |
| 7 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 8 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 9 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below. |

## EJBComponent (EJB Module)

## LibraryMessageBean.java

```
package com.tuturialspoint.messagebean;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

@MessageDriven(
   name = "BookMessageHandler",
   activationConfig = {
      @ActivationConfigProperty( propertyName = "destinationType",
                                 propertyValue = "javax.jms.Queue"
      @ActivationConfigProperty( propertyName = "destination",
                                 propertyValue ="/queue/BookQueue"
   }
)
public class LibraryMessageBean implements MessageListener {

   @Resource
   private MessageDrivenContext mdctx;

   @EJB
   LibraryPersistentBeanRemote libraryBean;

   public LibraryMessageBean(){
   }

   public void onMessage(Message message) {
      ObjectMessage objectMessage = null;
      try {
         objectMessage = (ObjectMessage) message;
         Book book = (Book) objectMessage.getObject();
         libraryBean.addBook(book);

      } catch (JMSException ex) {
         mdctx.setRollbackOnly();
      }
   }
}
```

## EJBTester (EJB Client)

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testMessageBeanEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
```

```
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testMessageBeanEjb(){

      try {
         int choice = 1;
         Queue queue = (Queue) ctx.lookup("/queue/BookQueue");
         QueueConnectionFactory factory =
         (QueueConnectionFactory) ctx.lookup("ConnectionFactory");
         QueueConnection connection =  factory.createQueueConnectio
         QueueSession session =
         connection.createQueueSession(false, QueueSession.AUTO_ACK
         QueueSender sender = session.createSender(queue);

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               ObjectMessage objectMessage =
                  session.createObjectMessage(book);
               sender.send(objectMessage);
            } else if (choice == 2) {
               break;
            }
         }

         LibraryPersistentBeanRemote libraryBean =
         (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
```

```
        }finally {
           try {
              if(brConsoleReader !=null){
                 brConsoleReader.close();
              }
           } catch (IOException ex) {
              System.out.println(ex.getMessage());
           }
        }
     }
  }
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- In testStatefulEjb() method, jndi lookup is done with name - "/queue/BookQueue" to obtain treference of queue available in Jboss. Then sender is created using queue session.
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and sender sends the book name to queue. When JBoss container receives this message in queue, it calls our message driven bean's onMessage method. Our message driven bean then saves book using stateful session bean addBook() method. Session Bean is persisting the book in database via EntityManager call.
- If user enters 2, then another jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful ejb) again and listing of books is done.

## 运行客户端访问EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
*********************
Welcome to Book Store
*********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn EJB
*********************
Welcome to Book Store
*********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 2
1\. learn java
1\. learn EJB
BUILD SUCCESSFUL (total time: 15 seconds)
```

上面显示的输出状态，我们的消息驱动bean接收消息和存储在持久性存储的书和书籍通过数据库检索。

# EJB注解/注释 - EJB

在Java 5.0中引入的注解。注释的目的是要重视在类中更多的信息或元数据在其源代码内类。在EJB 3.0中，注释是用来描述配置元数据在ejb类。通过这种方式，EJB3.0消除了需要描述在XML配置文件中的配置数据。

EJB容器使用的编译器工具来生成所需的工件，如接口，部署描述符，通过阅读这些注释。下面列出的常用的注解。

| Sr. No. | 名称 | 描述 |
| --- | --- | --- |
| 1 | javax.ejb.Stateless | 指定一个给定的EJB类是一个无状态会话bean. 属性 **name** - 用于指定会话bean的名称。 **mappedName** - 用于指定的会话bean的JNDI名称。 **description** - 用于提供会话bean的描述。 |
| 2 | javax.ejb.Stateful | 指定一个给定的EJB类是有状态会话bean。属性 **name** - 用于指定会话bean的名称。 **mappedName** - 用于指定的会话bean的JNDI名称。 **description** - 用于提供会话bean的描述。 |
| 3 | javax.ejb.MessageDrivenBean | 指定一个给定的EJB类是消息驱动的Bean。属性 **name** - 用于指定消息驱动bean的名称。 **messageListenerInterface** - 消息驱动bean用于指定消息监听器接口。 **activationConfig** - 用于指定的配置细节，消息驱动Bean消息驱动bean的经营环境。 **mappedName** - 用于指定的会话bean的JNDI名称。 **description** - 用于提供会话bean的描述。 |
| 4 | javax.ejb.EJB | 用于指定或注入到另一个EJB的EJB实例的依赖。属性 **name** - 用来指定将被使用的环境中找到引用的bean的名称。 **beanInterface** - 用于指定的接口类型引用的bean。 **beanName** - 用来提供引用的bean的名称。 **mappedName** - 用于引用bean指定的JNDI名称。 **description** - 用 |

| | | |
|---|---|---|
| | | 来提供引用的bean的描述。 |
| 5 | javax.ejb.Local | 用于指定一个会话bean的本地接口（次）。这个本地接口状态会话bean的业务方法（可以是无状态或有状态）。这个接口是用来以暴露本地客户端都运行在相同的部署/应用EJB的业务方法。 属性 **value** - 用于指定的本地接口列表接口数组。 |
| 6 | javax.ejb.Remote | 用于指定一个会话bean的远程接口（次）。这个远程接口状态会话bean的业务方法（可以是无状态或有状态）。这个接口是用来揭露远程客户端运行在不同的部署/应用EJB的业务方法。属性 **value** - 用于指定远程接口接口数组列表。 |
| 7 | javax.ejb.ActivationConfigProperty | 用于指定需要消息驱动的Bean的属性。例如终止点，目的地，消息选择等。这个注解通过作为参数，activationConfig属性javax.ejb.MessageDrivenBean注释。属性 **propertyName** - 属性名称. **propertyValue** - 属性值. |
| 8 | javax.ejb.PostActivate | 用于指定EJB的生命周期的回调方法。这种方法时，将调用EJB容器刚刚激活/激活bean实例。这个接口是用来以暴露本地客户端都运行在相同的部署/应用EJB的业务方法。 |

# EJB回调 - EJB

回调是一种机制，可以截获企业Bean的生命周期。 EJB 3.0规范指定的回调，回调处理方法是要创建。 EJB容器调用这些回调。 EJB类本身或在一个单独的类，我们可以定义回调方法。 EJB3.0提供了许多回调的注解

以下是无状态的bean回调的注解。

| Annotation | 描述 |
| --- | --- |
| @PostConstruct | method is invoked when a bean is created for the first time |
| --- | --- |
| @PreDestroy | method is invoked when a bean is removed from the bean pool or is destroyed. |
| --- | --- |

Following is the list of callback annotations for stateful bean.

| Annotation | 描述 |
| --- | --- |
| @PostConstruct | method is invoked when a bean is created for the first time |
| --- | --- |
| @PreDestroy | method is invoked when a bean is removed from the bean pool or is destroyed. |
| --- | --- |
| @PostActivate | method is invoked when a bean is loaded to be used. |
| --- | --- |
| @PrePassivate | method is invoked when a bean is put back to bean pool. |
| --- | --- |

Following is the list of callback annotations for message driven bean.

| Annotation | 描述 |
|---|---|
| @PostConstruct | method is invoked when a bean is created for the first time |
| --- | --- |
| @PreDestroy | method is invoked when a bean is removed from the bean pool or is destroyed. |
| --- | --- |

Following is the list of callback annotations for entity bean.

| Annotation | Description |
|---|---|
| @PrePersist | method is invoked when an entity is created in database. |
| --- | --- |
| @PostPersist | method is invoked after an entity is created in database. |
| --- | --- |
| @PreRemove | method is invoked when an entity is deleted from the database. |
| --- | --- |
| @PostRemove | method is invoked after an entity is deleted from the database. |
| --- | --- |
| @PreUpdate | method is invoked before an entity is to be updated in the database. |
| --- | --- |
| @PostLoad | method is invoked when a record is fetched from database and loaded into the entity. |
| --- | --- |

# 示例应用程序

让我们创建一个测试EJB应用程序来测试各种回调EJB。

| Step | 描述 |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.stateless* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Persistence* chapter as such for this chapter to add various callbacks to ejbs. |
| 2 | Create *LibrarySessionBean.java* and *LibrarySessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged. |
| 3 | Use Beans created in the *EJB - Persistence* chapter. Add callback methods as shown below. Keep rest of the files unchanged. |
| 4 | Create a java class *BookCallbackListener* under package *com.tutorialspoint.callback*. This class will demonstrates the seperation of callback methods. |
| 5 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 6 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 7 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

# EJBComponent (EJB Module)

# BookCallbackListener.java

```
package com.tutorialspoint.callback;

import javax.persistence.PrePersist;
import javax.persistence.PostLoad;
import javax.persistence.PostPersist;
import javax.persistence.PostRemove;
import javax.persistence.PostUpdate;
import javax.persistence.PreRemove;
import javax.persistence.PreUpdate;
import com.tutorialspoint.entity.Book;

public class BookCallbackListener {

   @PrePersist
```

```
    public void prePersist(Book book){
       System.out.println("BookCallbackListener.prePersist:"
          + "Book to be created with book id: "+book.getId());
    }

    @PostPersist
    public void postPersist(Object book){
       System.out.println("BookCallbackListener.postPersist::"
          + "Book created with book id: "+((Book)book).getId());
    }

    @PreRemove
    public void preRemove(Book book)
    {
       System.out.println("BookCallbackListener.preRemove:"
          + " About to delete Book: " + book.getId());
    }

    @PostRemove
    public void postRemove(Book book)
    {
       System.out.println("BookCallbackListener.postRemove::"
          + " Deleted Book: " + book.getId());
    }

    @PreUpdate
    public void preUpdate(Book book)
    {
       System.out.println("BookCallbackListener.preUpdate::"
          + " About to update Book: " + book.getId());
    }

    @PostUpdate
    public void postUpdate(Book book)
    {
       System.out.println("BookCallbackListener.postUpdate::"
          + " Updated Book: " + book.getId());
    }

    @PostLoad
    public void postLoad(Book book)
    {
       System.out.println("BookCallbackListener.postLoad::"
          + " Loaded Book: " + book.getId());
    }
}
```

## Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="books")
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

## LibraryStatefulSessionBean.java

```
package com.tutorialspoint.stateful;

import java.util.ArrayList;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
```

```
import javax.ejb.PostActivate;
import javax.ejb.PrePassivate;
import javax.ejb.Stateful;

@Stateful
public class LibraryStatefulSessionBean
   implements LibraryStatefulSessionBeanRemote {
   List<String> bookShelf;

   public LibraryStatefulSessionBean(){
      bookShelf = new ArrayList<String>();
   }

   public void addBook(String bookName) {
      bookShelf.add(bookName);
   }

   public List<String> getBooks() {
      return bookShelf;
   }

   @PostConstruct
   public void postConstruct(){
      System.out.println("LibraryStatefulSessionBean.postConstruct:
         + " bean created.");
   }

   @PreDestroy
   public void preDestroy(){
      System.out.println("LibraryStatefulSessionBean.preDestroy:"
         + " bean removed.");
   }

   @PostActivate
   public void postActivate(){
      System.out.println("LibraryStatefulSessionBean.postActivate:"
         + " bean activated.");
   }

   @PrePassivate
   public void prePassivate(){
      System.out.println("LibraryStatefulSessionBean.prePassivate:"
         + " bean passivated.");
   }
}
```

# LibraryStatefulSessionBeanRempote.java

```
package com.tutorialspoint.stateful;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryStatefulSessionBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

## LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean
   implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){}

   @PersistenceContext(unitName="EntityEjbPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book")
         .getResultList();
   }

   @PostConstruct
   public void postConstruct(){
      System.out.println("postConstruct:: LibraryPersistentBean ses
         + " created with entity Manager object: ");
   }

   @PreDestroy
   public void preDestroy(){
      System.out.println("preDestroy: LibraryPersistentBean session
      + " bean is removed ");
   }
}
```

## LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

- 只要你部署在JBoss EjbComponent项目，就会发现jboss的日志。
- JBoss已经自动为我们的会话bean创建一个JNDI条目 - **LibraryPersistentBean/remote**.
- 我们将使用这个查询字符串来获得远程类型的业务对象 - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.stateless.Librar
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
...
```

## EJBTester (EJB Client)

## jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- 这些属性是用来初始化InitialContext对象的Java命名服务

- InitialContext的对象将被用于查找无状态会话bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibrarySessionBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEntityEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
```

```
Enter Choice: ");
   }

   private void testEntityEjb(){
      try {
      int choice = 1;

      LibraryPersistentBeanRemote libraryBean =
      (LibraryPersistentBeanRemote)
      ctx.lookup("LibraryPersistentBean/remote");

      while (choice != 2) {
         String bookName;
         showGUI();
         String strChoice = brConsoleReader.readLine();
         choice = Integer.parseInt(strChoice);
         if (choice == 1) {
            System.out.print("Enter book name: ");
            bookName = brConsoleReader.readLine();
            Book book = new Book();
            book.setName(bookName);
            libraryBean.addBook(book);
         } else if (choice == 2) {
            break;
         }
      }

      List<Book> booksList = libraryBean.getBooks();

      System.out.println("Book(s) entered so far: " + booksList.siz
      int i = 0;
      for (Book book:booksList) {
         System.out.println((i+1)+". " + book.getName());
         i++;
      }

      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatelessEjb() method, jndi lookup is done with name - "LibrarySessionBean/remote" to obtain the remote business object (stateless ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in the database.
- If user enters 2, system retrieves books using stateless session bean getBooks() method and exits.

## 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择 **run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. Learn Java
BUILD SUCCESSFUL (total time: 13 seconds)
```

## JBoss应用服务器的日志输出

你可以找到以下的回调在JBoss日志项

```
14:08:34,293 INFO  [STDOUT] postConstruct:: LibraryPersistentBean s
...
16:39:09,484 INFO  [STDOUT] BookCallbackListener.prePersist:: Book 
16:39:09,531 INFO  [STDOUT] BookCallbackListener.postPersist:: Book
16:39:09,900 INFO  [STDOUT] BookCallbackListener.postLoad:: Loaded 
```

# EJB定时器服务 - EJB

定时器服务使用计划应用程序可以建立一个机制。例如，每月1日的工资单生成。EJB3.0规范指定超时注释，这有助于编程一个无状态或消息驱动Bean的EJB服务。EJB容器调用的方法，这是注释@Timeout.

EJB计时器服务是有助于创造的定时器，并安排回调计时器到期时由EJB容器提供的服务。

## 创建定时器的步骤

使用@ Resource注解注入SessionContext的bean

```
@Stateless
public class TimerSessionBean {

   @Resource
   private SessionContext context;
   ...
}
```

使用SessionContext对象TimerService创造定时器的。传递时间（以毫秒为单位）和消息。

```
public void createTimer(long duration) {
   context.getTimerService().createTimer(duration, "Hello World!")
}
```

## 使用定时器的步骤

使用@Timeout批注的方法。返回类型必须为void，并传递一个参数类型的定时器。我们取消计时器后第一次执行，否则将继续运行，修正后的时间间隔。

```
@Timeout
public void timeOutHandler(Timer timer){
   System.out.println("timeoutHandler : " + timer.getInfo());
   timer.cancel();
}
```

## 示例应用程序

让我们创建一个测试测试计时器服务在EJB的EJB应用程序中。

| Step | 描述 |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.timer* as explained in the *EJB - Create Application* chapter. |
| 2 | Create *TimerSessionBean.java* and *TimerSessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged. |
| 3 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 4 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 5 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

## EJBComponent (EJB Module)

## TimerSessionBean.java

```
package com.tutorialspoint.timer;

import javax.annotation.Resource;
import javax.ejb.SessionContext;
import javax.ejb.Timer;
import javax.ejb.Stateless;
import javax.ejb.Timeout;

@Stateless
public class TimerSessionBean implements TimerSessionBeanRemote {

   @Resource
   private SessionContext context;

   public void createTimer(long duration) {
      context.getTimerService().createTimer(duration, "Hello World
   }

   @Timeout
   public void timeOutHandler(Timer timer){
      System.out.println("timeoutHandler : " + timer.getInfo());
      timer.cancel();
   }
}
```

## TimerSessionBeanRemote.java

```
package com.tutorialspoint.timer;

import javax.ejb.Remote;

@Remote
public interface TimerSessionBeanRemote {
   public void createTimer(long milliseconds);
}
```

- 一旦你在Jboss应用服务器部署EjbComponent项目，发现jboss日志。
- JBoss已经自动为我们的会话bean创建一个JNDI条目 - **TimerSessionBean/remote**.
- 我们将使用这个查询字符串来获得远程类型的业务对象 - **com.tutorialspoint.timer.TimerSessionBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   TimerSessionBean/remote - EJB3.x Default Remote Business Interfa
   TimerSessionBean/remote-com.tutorialspoint.timer.TimerSessionBea
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.t
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- 这些属性是用来初始化InitialContext对象的Java命名服务
- InitialContext对象将被用于查找无状态会话bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.TimerSessionBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
```

```
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testTimerService();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testTimerService(){
      try {
         TimerSessionBeanRemote timerServiceBean = (TimerSessionBea

         System.out.println("["+(new Date()).toString()+ "]" + "tir
         timerServiceBean.createTimer(2000);

      } catch (NamingException ex) {
         ex.printStackTrace();
      }
   }
}
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- 在testTimerService（）方法，名字完成JNDI查找 - “TimerSessionBean/remote”，以获得远程业务对象（定时器无状态EJB）。
- 然后的调用createTimer通过预定时间为2000毫秒。

- 2秒后EJB容器调用timeoutHandler，方法。

# 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
[Wed Jun 19 11:35:47 IST 2013]timer created.
BUILD SUCCESSFUL (total time: 0 seconds)
```

# JBoss应用服务器的日志输出

你可以找到以下的回调在JBoss日志项

```
...
11:35:49,555 INFO  [STDOUT] timeoutHandler : Hello World!
...
```

# EJB依赖注入 - EJB

EJB 3.0规范提供了注释字段或setter方法注入依赖可以应用。 EJB容器使用的全局JNDI注册表定位的依赖。以下注解在EJB 3.0中使用依赖注入。

- **@EJB** - 用来注入其他EJB引用。
- **@Resource** - 用于注入数据源或单例服务，如sessionContext，timerService等

## @EJB使用步骤

@EJB 可以使用栏位或以下方式的方法。

```
public class LibraryMessageBean implements MessageListener {
   //dependency injection on field.
   @EJB
   LibraryPersistentBeanRemote libraryBean;
   ...
}
```

```
public class LibraryMessageBean implements MessageListener {

   LibraryPersistentBeanRemote libraryBean;

   //dependency injection on method.
   @EJB(beanName="com.tutorialspoint.stateless.LibraryPersistentBea
   public void setLibraryPersistentBean(
   LibraryPersistentBeanRemote libraryBean)
   {
      this.libraryBean = libraryBean;
   }
   ...
}
```

## @Resource 使用步骤

@Resource 通常用于注入EJB容器提供单例。

```
public class LibraryMessageBean implements MessageListener {
   @Resource
   private MessageDrivenContext mdctx;
   ...
}
```

# 示例应用程序

让我们创建一个测试EJB应用程序来测试EJB服务的依赖注入。

| Step | Description |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.timer* as explained in the *EJB - Create Application* chapter. |
| 3 | Use Beans created in the *EJB - Message Driven Bean* chapter. Keep rest of the files unchanged. |
| 5 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 6 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 7 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

## EJBComponent (EJB Module)

## LibraryMessageBean.java

```
package com.tuturialspoint.messagebean;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

@MessageDriven(
   name = "BookMessageHandler",
   activationConfig = {
      @ActivationConfigProperty( propertyName = "destinationType",
                                 propertyValue = "javax.jms.Queue"
      @ActivationConfigProperty( propertyName = "destination",
                                 propertyValue ="/queue/BookQueue"
   }
)
public class LibraryMessageBean implements MessageListener {

   @Resource
   private MessageDrivenContext mdctx;

   @EJB
   LibraryPersistentBeanRemote libraryBean;

   public LibraryMessageBean(){
   }

   public void onMessage(Message message) {
      ObjectMessage objectMessage = null;
      try {
         objectMessage = (ObjectMessage) message;
         Book book = (Book) objectMessage.getObject();
         libraryBean.addBook(book);

      } catch (JMSException ex) {
         mdctx.setRollbackOnly();
      }
   }
}
```

## EJBTester (EJB Client)

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testMessageBeanEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
```

```
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testMessageBeanEjb(){

      try {
         int choice = 1;
         Queue queue = (Queue) ctx.lookup("/queue/BookQueue");
         QueueConnectionFactory factory =
         (QueueConnectionFactory) ctx.lookup("ConnectionFactory");
         QueueConnection connection =  factory.createQueueConnectio
         QueueSession session = connection.createQueueSession(
         false, QueueSession.AUTO_ACKNOWLEDGE);
         QueueSender sender = session.createSender(queue);

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               ObjectMessage objectMessage =
               session.createObjectMessage(book);
               sender.send(objectMessage);
            } else if (choice == 2) {
               break;
            }
         }

         LibraryPersistentBeanRemote libraryBean =
         (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: "
         + booksList.size());
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
```

```
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- JNDI查找testStatefulEjb（）方法，完成了名字 - “/queue/BookQueue”在JBoss获得参考。发件人使用队列会话创建。
- 然后用户显示一个库存储的用户界面和他/她被要求输入选择。
- 如果用户输入1，系统要求输入书籍名称和发件人发送本书的名字来排队。当JBoss容器接收到这个消息队列，它会调用我们的消息驱动bean的onMessage方法。消息驱动bean保存使用状态会话bean addBook（）方法的书。会话Bean持久化这本书中通过EntityManager调用数据库。
- 如果用户输入2，然后是另一个JNDI查找名字 - “LibraryStatefulSessionBean/remote”获取远程业务对象（状态EJB）再次上市的书籍完成。

# 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn EJB
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 2
1\. learn java
1\. learn EJB
BUILD SUCCESSFUL (total time: 15 seconds)
```

- 输出上面显示说我们的消息驱动bean接收消息和持久性存储和书籍从数据库中检索存储本书。
- 我们的消息驱动bean被使用LibraryPersistentBean使用@ EJB注释注入的情况下例外MessageDrivenContext对象用于回滚事务。

# EJB拦截 - EJB

EJB3.0提供了拦截业务方法的规范使用@ AroundInvoke注释注释的方法调用。一个拦截器方法被称为包含EJBContainer的业务方法调用前拦截。下面的例子是一个拦截器方法签名

```
@AroundInvoke
public Object methodInterceptor(InvocationContext ctx) throws Excep
{
   System.out.println("*** Intercepting call to LibraryBean method
   + ctx.getMethod().getName());
   return ctx.proceed();
}
```

拦截器方法可以应用在三个层面上的约束

- **Default** - 默认的拦截器被调用内deployment.Default拦截每个bean只能应用于通过XML（ejb-jar.xml）。
- **Class** - 类级别拦截所有的bean的方法被调用。类级别拦截器可以应用通过XML注释（ejb-jar.xml）。
- **Method** - 方法级别的拦截器是一个特定的bean的方法调用。方法级的拦截器可以应用通过XML注释（ejb-jar.xml）。

我们在这里讨论类级别拦截。

*Interceptor class*

```
package com.tutorialspoint.interceptor;

import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

public class BusinessInterceptor {
   @AroundInvoke
   public Object methodInterceptor(InvocationContext ctx) throws Ex
   {
      System.out.println("*** Intercepting call to LibraryBean meth
      + ctx.getMethod().getName());
      return ctx.proceed();
   }
}
```

*Remote Interface*

```
import javax.ejb.Remote;

@Remote
public interface LibraryBeanRemote {
   //add business method declarations
}
```

*Intercepted Stateless EJB*

```
@Interceptors ({BusinessInterceptor.class})
@Stateless
public class LibraryBean implements LibraryBeanRemote {
   //implement business method
}
```

# 示例应用程序

让我们创建一个测试测试截获的无状态EJB的EJB应用程序。

| Step | 描述 |
| --- | --- |
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.interceptor_as explained in the _EJB - Create Application* chapter. You can also use the project created in*EJB - Create Application* chapter as such for this chapter to understand intercepted ejb concepts. |
| 2 | Create *LibraryBean.java* and *LibraryBeanRemote* under package*com.tutorialspoint.interceptor* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged. |
| 3 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 4 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 5 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

# EJBComponent (EJB Module)

# LibraryBeanRemote.java

```
package com.tutorialspoint.interceptor;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

## LibraryBean.java

```
package com.tutorialspoint.interceptor;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;
import javax.interceptor.Interceptors;

@Interceptors ({BusinessInterceptor.class})
@Stateless
public class LibraryBean implements LibraryBeanRemote {

   List<String> bookShelf;

   public LibraryBean(){
      bookShelf = new ArrayList<String>();
   }

   public void addBook(String bookName) {
      bookShelf.add(bookName);
   }

   public List<String> getBooks() {
      return bookShelf;
   }
}
```

- 只要你部署在JBoss EjbComponent项目，会注意到jboss的日志。
- JBoss已经自动为我们的会话bean创建一个JNDI条目 - **LibraryBean/remote**.
- 我们将使用这个查询字符串来获得远程类型的业务对象 - **com.tutorialspoint.interceptor.LibraryBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibraryBean/remote - EJB3.x Default Remote Business Interface
   LibraryBean/remote-com.tutorialspoint.interceptor.LibraryBeanRen
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.i
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibraryBean/remote - EJB3.x Default Remote Business Interface
   LibraryBean/remote-com.tutorialspoint.interceptor.LibraryBeanRen
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- 这些属性是用来初始化InitialContext对象的Java命名服务
- InitialContext的对象将被用于查找无状态会话bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
```

```
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testInterceptedEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testInterceptedEjb(){

      try {
         int choice = 1;

         LibraryBeanRemote libraryBean =
         LibraryBeanRemote)ctx.lookup("LibraryBean/remote");

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
            } else if (choice == 2) {
```

```
                break;
             }
          }

          List<Book> booksList = libraryBean.getBooks();

          System.out.println("Book(s) entered so far: " + booksList.
          int i = 0;
          for (Book book:booksList) {
             System.out.println((i+1)+". " + book.getName());
             i++;
          }
       } catch (Exception e) {
          System.out.println(e.getMessage());
          e.printStackTrace();
       }finally {
          try {
             if(brConsoleReader !=null){
                brConsoleReader.close();
             }
          } catch (IOException ex) {
             System.out.println(ex.getMessage());
          }
       }
    }
}
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- JNDI查找testInterceptedEjb（）方法，完成了名称 - “LibraryBean/remote”获得远程业务对象（无状态EJB）。
- 如果用户输入1，系统要求输入书籍名称和节约使用无状态的会话bean addBook（）方法的书。会话Bean存储在它的实例变量的书。
- 如果用户输入2，系统检索书使用状态会话Bean getBooks（）方法和退出。

# Run Client to access EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. Learn Java
BUILD SUCCESSFUL (total time: 13 seconds)
```

## JBoss应用服务器的日志输出

验证下面的输出在JBoss应用服务器的日志输出。

```
....
09:55:40,741 INFO  [STDOUT] *** Intercepting call to LibraryBean me
09:55:43,661 INFO  [STDOUT] *** Intercepting call to LibraryBean me
```

# EJB嵌入对象 - EJB

EJB 3.0中提供了选项嵌入到实体bean的Java POJO（普通Java对象）和嵌入式POJO类的方法允许列名映射。嵌入一个java的POJO必须定义了@ Embeddable注解。

```
@Embeddable
public class Publisher implements Serializable{
   private String name;
   private String address;
   ...
}
```

上面的类可以使用@ Embedded批注嵌入

```
@Entity
public class Book implements Serializable{
   private int id;
   private String name;
   private Publisher publisher;
   ...
   @Embedded
   @AttributeOverrides({
   @AttributeOverride(name = "name",
                      column = @Column(name = "PUBLISHER")),
   @AttributeOverride(name = "address",
                      column = @Column(name = "PUBLISHER_ADDRESS"))
   })
   public Publisher getPublisher() {
      return publisher;
   }
   ...
}
```

## 示例应用程序

让我们创建一个测试EJB应用程序来测试EJB 3.0中的嵌入对象。

| Step | 描述 |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Please use the project created in *EJB - Persistence* chapter as such for this chapter to understand embedded objects in ejb concepts. |
| 2 | Create *Publisher.java* under package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged. |
| 3 | Create *Book.java* under package *com.tutorialspoint.entity*. Use *EJB - Persistence* chapter as reference. Keep rest of the files unchanged. |
| 4 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 5 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 6 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

## Create/Alter book table

```
CREATE TABLE book (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
Alter table book add publisher varchar(100);
Alter table book add publisher_address varchar(200);
```

## EJBComponent (EJB Module)

## Publisher.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;
import javax.persistence.Embeddable;

@Embeddable
public class Publisher implements Serializable{

   private String name;
   private String address;

   public Publisher(){}

   public Publisher(String name, String address){
      this.name = name;
      this.address = address;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getAddress() {
      return address;
   }

   public void setAddress(String address) {
      this.address = address;
   }

   public String toString(){
      return name + "," + address;
   }
}
```

# Book.java

```
package com.tutorialspoint.entity;

import com.tutorialspoint.callback.BookCallbackListener;
import java.io.Serializable;
import javax.persistence.AttributeOverride;
import javax.persistence.AttributeOverrides;
import javax.persistence.Column;
```

```
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="book")
public class Book implements Serializable{

   private int id;
   private String name;
   private Publisher publisher;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   @Embedded
   @AttributeOverrides({
      @AttributeOverride(name = "name",
         column = @Column(name = "PUBLISHER")),
      @AttributeOverride(name = "address",
         column = @Column(name = "PUBLISHER_ADDRESS"))
   })
   public Publisher getPublisher() {
      return publisher;
   }

   public void setPublisher(Publisher publisher) {
      this.publisher = publisher;
   }
}
```

## LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

## LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBean

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList(
   }
}
```

- 一旦你部署EjbComponent项目在Jboss应用服务器，发现jboss的日志。
- JBoss已经自动为我们的会话bean创建一个JNDI入口 - **LibraryPersistentBean/remote**.
- 我们将使用这个查询字符串来获得远程类型的业务对象 - **com.tutorialspoint.interceptor.LibraryPersistentBeanRemote**

# JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.interceptor.Libr
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.i
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.interceptor.Libr
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- 这些属性是用来初始化InitialContext对象的Java命名服务
- InitialContext的对象将被用于查找无状态会话bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEmbeddedObjects();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testEmbeddedObjects(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
        (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");
```

```
         while (choice != 2) {
            String bookName;
            String publisherName;
            String publisherAddress;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               System.out.print("Enter publisher name: ");
               publisherName = brConsoleReader.readLine();
               System.out.print("Enter publisher address: ");
               publisherAddress = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               book.setPublisher
              (new Publisher(publisherName,publisherAddress));

               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            System.out.println("Publication: "+book.getPublisher()
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- In testInterceptedEjb() method, jndi lookup is done with name - "LibraryPersistenceBean/remote" to obtain the remote business object (stateless ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in database.
- If user enters 2, system retrives books using stateless session bean getBooks() method and exits.

## 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择 **run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: learn html5
Enter publisher name: SAMS
Enter publisher address: DELHI
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. learn html5
Publication: SAMS,DELHI
BUILD SUCCESSFUL (total time: 21 seconds)
```

# EJB - Blobs/Clobs - EJB

EJB3.0提供支持BLOB和CLOB类型，使用@Lob注解。下面的Java类型可以映射使用@Lob注解。

- java.sql.Blob
- java.sql.Clob
- byte[]
- String
- Serializable Object

```
@Entity
@Table(name="books")
@EntityListeners(BookCallbackListener.class)
public class Book implements Serializable{
   ...
   private byte[] image;

   @Lob @Basic(fetch= FetchType.EAGER)
   public byte[] getImage() {
      return image;
   }
   ...
}
```

## 示例应用程序

让我们创建一个测试EJB在EJB3.0应用程序来测试BLOB / CLOB支持。

| Step | 描述 |
| --- | --- |
| 1 | 创建一个项目与一个名字EjbComponent包com.youcompany.entity下EJB中的解释 - 创建应用程序的章节。请作为的项目中创建EJB -阅读持久性本章可了解CLOB/ BLOB对象EJB概念。 |
| 2 | 创建Book.java包com.youcompany.entity下。使用EJB - 持久性章作为参考。其余文件保持不变。 |
| 3 | 清理并生成应用程序以确保业务逻辑是按要求工作。 |
| 4 | 最后，将应用程序部署在JBoss应用服务器上的jar文件的形式。 JBoss应用服务器将自动开始浏览网页，如果它尚未启动。 |
| 5 | 现在创建EJB客户端，基于控制台的应用程序以同样的方式在EJB解释 - 下创建应用程序本章主题 **Create Client to access EJB**. |

## Create/Alter book table

```
CREATE TABLE book (
   id      integer PRIMARY KEY,
   name    varchar(50)
);
Alter table book add image bytea;
Alter table book add xml text;
```

## EJBComponent (EJB Module)

## Book.java

```
package com.tutorialspoint.entity;

import com.tutorialspoint.callback.BookCallbackListener;
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;
```

```
@Entity
@Table(name="book")
public class Book implements Serializable{

   private int id;
   private String name;
   private byte[] image;
   private String xml;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   @Lob @Basic(fetch= FetchType.EAGER)
   public byte[] getImage() {
      return image;
   }

   public void setImage(byte[] image) {
      this.image = image;
   }

   @Lob @Basic(fetch= FetchType.EAGER)
   public String getXml() {
      return xml;
   }

   public void setXml(String xml) {
      this.xml = xml;
   }
}
```

## LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

## LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBear

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList(
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.

- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We'll using this lookup string to get remote business object of type - **com.tutorialspoint.interceptor.LibraryPersistentBeanRemote**

## JBoss Application server log output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.interceptor.Libr
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.i
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.interceptor.Libr
...
```

## EJBTester (EJB Client)

## jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service
- InitialContext object will be used to lookup stateless session bean

## EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testBlobClob();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testBlobClob(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
        (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");
```

```
         while (choice != 2) {
            String bookName;
            String publisherName;
            String publisherAddress;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               String xml = "<book><name>"+bookName+"</name></book>
               Book book = new Book();
               book.setName(bookName);
               byte[] imageBytes = {0x32, 0x32,0x32, 0x32,0x32,
               0x32,0x32, 0x32,
               0x32, 0x32,0x32, 0x32,0x32, 0x32,0x32, 0x32,
               0x32, 0x32,0x32, 0x32,0x32, 0x32,0x32, 0x32
               };
               book.setImage(imageBytes);
               book.setXml(xml);

               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            byte[] imageByts = book.getImage();
            if(imageByts != null){
               System.out.print("image bytes: [");
               for(int j = 0; j < imageByts.length ; j++){
                  System.out.print("0x"
                  + String.format("%x", imageByts[j]) +" ");
               }
               System.out.println("]");
            }
            System.out.println(book.getXml());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
```

```
            } catch (IOException ex) {
               System.out.println(ex.getMessage());
            }
         }
      }
   }
```

EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testInterceptedEjb() method, jndi lookup is done with name - "LibraryPersistenceBean/remote" to obtain the remote business object (stateless ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in database.
- If user enters 2, system retrieves books using stateless session bean getBooks() method and exits.

# Run Client to access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
********************
Welcome to Book Store
********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: learn testing
********************
Welcome to Book Store
********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. learn testing
image bytes: [
0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x
<book><name>learn testing</name></book>
BUILD SUCCESSFUL (total time: 20 seconds)
```

# EJB事务 - EJB

事务是一个单元的工作项目遵循ACID特性。ACID代表原子性，一致性，独立性和持久性。

- 原子 -如果有任何的工作项失败，完整的单元被认为是失败。成功意味着所有项目的成功执行。
- 一致性- 事务必须保持系统处于一致状态。
- 独立性- 每个事务执行独立的任何其他的事务。
- 持久性 - 事务应该生存系统故障，如果已经执行或提交。

EJB容器/服务器，的事务服务器和处理事务上下文的传递和分布式事务。事务可以由容器管理或bean的代码自定义代码处理。

- **Container Managed Transactions** - 在这种类型中，容器管理的事务状态。
- **Bean Managed Transactions** - 在这种类型中，开发者管理事务状态的生命周期。

## 容器管理的事务

指定EJB3.0，EJB容器实现的事务特性。

- **REQUIRED** - 表示业务方法已被执行的范围内的事务，否则将开始一个新的事务方法。
- **REQUIRES_NEW** - 表示要开始一个新的事务，业务方法。
- **SUPPORTS** - 表示业务方法将执行作为的事务的一部分。
- **NOT_SUPPORTED** - 表示业务方法不应该被执行作为的事务的一部分。
- **MANDATORY** - 表示业务方法将执行作为的事务的一部分，否则将引发异常。
- **NEVER** - 表示如果业务方法执行作为的事务的一部分，那么会抛出一个异常。

## 例子

```
package com.tutorialspoint.txn.required;

import javax.ejb.*

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class UserDetailBean implements UserDetailRemote {

   private UserDetail;

   @TransactionAttribute(TransactionAttributeType.REQUIRED)
   public void createUserDetail() {
      //create user details object
   }
}
```

createUserDetail（）业务方法时，需要使用Required注解。

```
package com.tutorialspoint.txn.required;

import javax.ejb.*

@Stateless
public class UserSessionBean implements UserRemote {

   private User;

   @EJB
   private UserDetailRemote userDetail;

   public void createUser() {
      //create user
      //...
      //create user details
      userDetail.createUserDetail();
   }
}
```

为createUser（）业务方法的使用createUserDetail（）。如果为createUser（期间发生的异常）调用和用户对象不会创建那么UserDetail对象也不会被创建。

# Bean管理的事务

Bean管理的事务，事务处理应用水平的异常，可以管理。以下是要考虑的关键点

- **Start** - 当启动一个事务中的业务方法。
- **Sucess** - 确定成功的情况下，当一个事务被提交。

- **Failed** - 确定失败的情况下，事务回滚。

# 实例

```
package com.tutorialspoint.txn.bmt;

import javax.annotation.Resource;
import javax.ejb.Stateless;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;
import javax.transaction.UserTransaction;

@Stateless
@TransactionManagement(value=TransactionManagementType.BEAN)
public class AccountBean implements AccountBeanLocal {

   @Resource
   private UserTransaction userTransaction;

   public void transferFund(Account fromAccount, double fund ,
      Account toAccount) throws Exception{

      try{
         userTransaction.begin();

         confirmAccountDetail(fromAccount);
         withdrawAmount(fromAccount,fund);

         confirmAccountDetail(toAccount);
         depositAmount(toAccount,fund);

         userTransaction.commit();
      }catch (InvalidAccountException exception){
         userTransaction.rollback();
      }catch (InsufficientFundException exception){
         userTransaction.rollback();
      }catch (PaymentException exception){
         userTransaction.rollback();
      }
   }

   private void confirmAccountDetail(Account account)
      throws InvalidAccountException {
   }

   private void withdrawAmount() throws InsufficientFundException {
   }

   private void depositAmount() throws PaymentException{
   }
}
```

在这个例子中，我们使用UserTransaction接口标记开始使用userTransaction.begin（）方法调用事务处理。我们标记事务完成使用userTransaction.commit（）方法，如果期间发生任何异常，那么我们完整的事务回滚使用userTransaction.rollback（）方法调用。

# EJB安全 - EJB

安全性是任何企业级应用关注的重大问题。它包括用户身份识别（S）或系统访问的应用程序，并允许或拒绝的访问应用程序内的资源。在EJB中，安全性可以声明的方式称为声明性安全EJB容器管理的安全问题，或者自定义代码可以在EJB处理安全问题的关注，通过自己声明。

## 安全的重要术语

- 认证 - 这是确保用户访问系统或应用程序被验证为是正品的方法。
- 授权 - 这是过程，确保用户有权访问系统资源的权限级别。
- 用户 - 用户表示访问该应用程序的客户端或系统。
- 用户组 - 用户可能具有一定部门例如管理员组的一部分。
- 用户角色 - 角色定义授权用户有权限访问系统资源的水平。

## 容器管理安全

EJB3.0指定下列属性/安全EJB容器实现的注解。

- **DeclareRoles** - 指示类将接受这些声明的的角色。注释在类级别应用。
- **RolesAllowed** -指示指定的角色的用户可以访问的方法。可以应用在一流水平，导致指定角色的用户可以访问的所有类方法。
- **PermitAll** - 表示该业务的方法是访问。可以应用在类以及方法级别。
- **DenyAll** - 表明业务方法不能访问任何用户指定的类或方法级别。

## 实例

```
package com.tutorialspoint.security.required;

import javax.ejb.*

@Stateless
@DeclareRoles({"student" "librarian"})
public class LibraryBean implements LibraryRemote {

   @RolesAllowed({"librarian"})
   public void delete(Book book){
      //delete book
   }

   @PermitAll
   public void viewBook(Book book){
      //view book
   }

   @DenyAll
   public void deleteAll(){
      //delete all books
   }
}
```

## 安全配置

在配置文件中的的映射角色和用户组。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Applic
<ejb-jar>
   <security-role-mapping>
      <role-name>student</role-name>
      <group-name>student-group</group-name>
   </security-role-mapping>
   <security-role-mapping>
      <role-name>librarian</role-name>
      <group-name>librarian-group</group-name>
   </security-role-mapping>
   <enterprise-beans/>
</ejb-jar>
```

# EJBJNDI绑定 - EJB

JNDI代表Java命名和目录接口。它是一组API和服务接口。基于Java的应用程序使用JNDI命名和目录服务。在EJB的背景下，有两个方面。

- **Binding** - 这指的是以后可以使用一个EJB对象分配一个名称。
- **Lookup** - 这指的是寻找并获得EJB对象。

在JBoss中，会话bean绑定到JNDI，默认情况下有以下格式。

- **local** - ejb-name/local
- **remote** - ejb-name/remote

情况下，EJB捆绑在一起<application-name> ear文件默认格式如下。

- **local** - application-name/ejb-name/local
- **remote** - application-name/ejb-name/remote

## 默认绑定的例子

请参阅EJB - 创建应用本章的JBoss的控制台输出。

JBoss应用服务器的日志输出

```
...
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibrarySessionBean/remote - EJB3.x Default Remote Business Inte
   LibrarySessionBean/remote-com.tutorialspoint.stateless.LibrarySe
...
```

## 定制绑定

以下注释可以用来定制默认JNDI绑定。

- **local** - org.jboss.ejb3.LocalBinding
- **remote** - org.jboss.ejb3.RemoteBindings

更新LibrarySessionBean.java。请参阅EJB - 创建应用程序一章

*LibrarySessionBean*

```
package com.tutorialspoint.stateless;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;

@Stateless
@LocalBinding(jndiBinding="tutorialsPoint/librarySession")
public class LibrarySessionBean implements LibrarySessionBeanLocal

    List<String> bookShelf;

    public LibrarySessionBean(){
       bookShelf = new ArrayList<String>();
    }

    public void addBook(String bookName) {
       bookShelf.add(bookName);
    }

    public List<String> getBooks() {
        return bookShelf;
    }
}
```

*LibrarySessionBeanLocal*

```
package com.tutorialspoint.stateless;

import java.util.List;
import javax.ejb.Local;

@Local
public interface LibrarySessionBeanLocal {

    void addBook(String bookName);

    List getBooks();

}
```

构建项目。将应用程序部署在JBoss在JBoss控制台验证下面的输出。

```
...
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   tutorialsPoint/librarySession - EJB3.x Default Local Business In
   tutorialsPoint/librarySession-com.tutorialspoint.stateless.Libra
...
```

重复上述步骤，为远程和检查结果。

# EJB实体关系 - EJB

EJB 3.0中提供的选项来定义像一对一的数据库实体关系/映射，一对多，多对一和多对多关系。以下是相关的注释。

- **OneToOne** - 对象都具有一对一的关系。例如，乘客可以在时间旅行使用一张票。
- **OneToMany** - 对象是具有一对多的关系。例如，一个父亲可以有多个孩子。
- **ManyToOne** - 对象有多对一的关系。举例来说，多个孩子对一个母亲。
- **ManyToMany** - 对象是多对多的关系。举例来说，一本书可以多发作者，一个作者可以写多本书。

在这里，我们将演示如何使用多对多的映射。要代表多对多的关系，三表是必需的。

- **Book** - 书籍记录表
- **Author** - 作者Author表记录
- **Book_Author** - BOOK_AUTHOR上述Book和Author表的表具有关联。

## 创建表

创建表**book author**, **book_author** 在默认数据库 **postgres**.

```
CREATE TABLE book (
   book_id     integer,
   name   varchar(50)
);
```

```
CREATE TABLE author (
   author_id   integer,
   name   varchar(50)
);
```

```
CREATE TABLE book_author (
   book_id     integer,
   author_id   integer
);
```

# 创建实体类

```
@Entity
@Table(name="author")
public class Author implements Serializable{
   private int id;
   private String name;
   ...
}
```

```
@Entity
@Table(name="book")
public class Book implements Serializable{
   private int id;
   private String title;
   private Set<Author> authors;
   ...
}
```

Use ManyToMany annotation in Book Entity

```
@Entity
public class Book implements Serializable{
   ...
   @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}
      , fetch = FetchType.EAGER)
   @JoinTable(table = @Table(name = "book_author"),
      joinColumns = {@JoinColumn(name = "book_id")},
      inverseJoinColumns = {@JoinColumn(name = "author_id")})
   public Set<Author> getAuthors()
   {
      return authors;
   }
   ...
}
```

# 实例应用

让我们创建一个测试EJB应用程序来测试EJB3.0实体关系对象。

| Step | 描述 |
|---|---|
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Please use the project created in *EJB - Persistence* chapter as such for this chapter to understand embedded objects in ejb concepts. |
| 2 | Create *Author.java* under package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged. |
| 3 | Create *Book.java* under package *com.tutorialspoint.entity*. Use *EJB - Persistence* chapter as reference. Keep rest of the files unchanged. |
| 4 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 5 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 6 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. |

## EJBComponent (EJB Module)

## Author.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="author")
public class Author implements Serializable{

   private int id;
   private String name;

   public Author(){}

   public Author(int id, String name){
      this.id = id;
      this.name = name;
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="author_id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String toString(){
      return id + "," + name;
   }
}
```

## Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;

@Entity
@Table(name="book")
public class Book implements Serializable{

   private int id;
   private String name;
   private Set<Author> authors;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="book_id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public void setAuthors(Set<Author> authors) {
      this.authors = authors;
   }

   @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}
      , fetch = FetchType.EAGER)
   @JoinTable(table = @Table(name = "book_author"),
      joinColumns = {@JoinColumn(name = "book_id")},
      inverseJoinColumns = {@JoinColumn(name = "author_id")})
   public Set<Author> getAuthors()
   {
```

```
        return authors;
    }
}
```

## LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

## LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBear

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList(
   }
}
```

- 只要你在JBoss部署 EjbComponent项目，会注意到jboss的日志。
- JBoss已经自动为我们的会话bean创建一个JNDI条目 - **LibraryPersistentBean/remote**.
- 我们将使用这个查询字符串来获得远程类型的业务对象 - **com.tutorialspoint.interceptor.LibraryPersistentBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the followin
   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.interceptor.Libr
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.i
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the followin

   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.interceptor.Libr
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service
- InitialContext object will be used to lookup stateless session bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
```

```
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEmbeddedObjects();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testEmbeddedObjects(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
         (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
            String authorName;

            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               System.out.print("Enter author name: ");
```

```
                authorName = brConsoleReader.readLine();
                Book book = new Book();
                book.setName(bookName);
                Author author = new Author();
                author.setName(authorName);
                Set<Author> authors = new HashSet<Author>();
                authors.add(author);
                book.setAuthors(authors);

                libraryBean.addBook(book);
             } else if (choice == 2) {
                break;
             }
          }

          List<Book> booksList = libraryBean.getBooks();

          System.out.println("Book(s) entered so far: " + booksList.
          int i = 0;
          for (Book book:booksList) {
             System.out.println((i+1)+". " + book.getName());
             System.out.print("Author: ");
             Author[] authors = (Author[])books.getAuthors().toArray
             for(int j=0;j<authors.length;j++){
                System.out.println(authors[j]);
             }
             i++;
          }
       } catch (Exception e) {
          System.out.println(e.getMessage());
          e.printStackTrace();
       }finally {
          try {
             if(brConsoleReader !=null){
                brConsoleReader.close();
             }
          } catch (IOException ex) {
             System.out.println(ex.getMessage());
          }
       }
    }
 }
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- 在testInterceptedEjb（）方法，JNDI查找名称 - 的“LibraryPersistenceBean/远程”获得远程业务对象（无状态的EJB）。
- 然后用户显示一个库存储的用户界面和他/她被要求输入选择。

- 如果用户输入1，系统要求输入书籍名称和节约使用无状态的会话bean addBook（）方法的书。会话Bean在数据库中存储的书。
- 如果用户输入2，系统检索书使用状态会话Bean getBooks（）方法和退出。

## 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: learn html5
Enter Author name: Robert
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. learn html5
Author: Robert
BUILD SUCCESSFUL (total time: 21 seconds)
```

# EJB访问数据库 - EJB

EJB 3.0持久性机制来访问容器管理的数据库中的数据库相关的操作。开发人员访问数据库可以直接使用JDBC API调用EJB的业务方法。

为了证明在ejb的数据库访问，我们要做好以下几项工作。

- 步骤 1. 在数据库中创建表.
- 步骤 2. 创建一个无状态EJB.
- 步骤 3. 更新无状态的EJB。添加添加记录并获得通过实体管理器从数据库中记录的方法。
- 步骤 4. 一个基于控制台应用程序客户端将访问无状态EJB的持久化数据库中的数据。

## 创建表

创建一张表 **books** 在默认的数据库 **postgres**.

```
CREATE TABLE books (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
```

## 创建模型类

```
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }
   ...
}
```

## 创建无状态EJB

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBean

   public void addBook(Book book) {
     //persist book using jdbc calls - by yiibai.com
   }

   public List<Book> getBooks() {
     //get books using jdbc calls
   }
   ...
}
```

构建EJB模块后，我们需要一个无状态的bean，我们将在下一节要创建客户端来访问。

## 实例应用

让我们创建一个测试EJB应用程序来测试EJB的数据库访问机制。

| Step | 描述 |
| --- | --- |
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand ejb data access concepts. |
| 2 | Create *Book.java* under package *com.tutorialspoint.entity* and modify it as shown below. |
| 3 | Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as explained in the *EJB - Create Application* chapter and modify them as shown below. |
| 4 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 5 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 6 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below. |

# EJBComponent (EJB Module)

## Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;

public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

## LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

# LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBear

   public LibraryPersistentBean(){
   }

   public void addBook(Book book) {
      Connection con = null;
      String url = "jdbc:postgresql://localhost:5432/postgres";
      String driver = "org.postgresql.driver";

      String userName = "sa";
      String password = "sa";
      List<Book> books = new ArrayList<Book>();
      try {

         Class.forName(driver).newInstance();
         con = DriverManager.getConnection(url , userName, passworc

         PreparedStatement st =
         con.prepareStatement("insert into book(name) values(?)");
         st.setString(1,book.getName());

         int result = st.executeUpdate();

      } catch (SQLException ex) {
         ex.printStackTrace();
      } catch (InstantiationException ex) {
         ex.printStackTrace();
      } catch (IllegalAccessException ex) {
         ex.printStackTrace();
      } catch (ClassNotFoundException ex) {
         ex.printStackTrace();
      }
   }
```

```
    public List<Book> getBooks() {
       Connection con = null;
       String url = "jdbc:postgresql://localhost:5432/postgres";
       String driver = "org.postgresql.driver";

       String userName = "sa";
       String password = "sa";
       List<Book> books = new ArrayList<Book>();
       try {

          Class.forName(driver).newInstance();
          con = DriverManager.getConnection(url , userName, passwor

          Statement st = con.createStatement();
          ResultSet rs = st.executeQuery("select * from book");

          Book book;
          while (rs.next()) {
             book = new Book();
             book.setId(rs.getInt(1));
             book.setName(rs.getString(2));
             books.add(book);
          }
       } catch (SQLException ex) {
          ex.printStackTrace();
       } catch (InstantiationException ex) {
          ex.printStackTrace();
       } catch (IllegalAccessException ex) {
          ex.printStackTrace();
       } catch (ClassNotFoundException ex) {
          ex.printStackTrace();
       }
       return books;
    }
 }
```

- 只要你在JBoss部署 EjbComponent项目，注意到jboss的日志。
- JBoss已经自动为我们的会话bean创建一个JNDI条目 - **LibraryPersistentBean/remote**.
- 我们将使用这个查询字符串来获得远程类型的业务对象 - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

# JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.stateless.Librar
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.stateless.Librar
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- 这些属性是用来初始化InitialContext对象的Java命名服务
- InitialContext的对象将被用于查找无状态会话bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
```

```
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEntityEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testEntityEjb(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
         LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
```

```
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- 在testStatefulEjb（）方法，JNDI名称进行查找 - "LibraryStatelessSessionBean/remote"以获得远程业务对象（状态EJB）。
- 然后用户显示一个库存储的用户界面和他/她被要求输入选择。
- 如果用户输入1，系统要求输入书籍名称和可以节省使用无状态的会话bean addBook（）方法。会话Bean坚持这本书中通过EntityManager的调用数据库。
- 如果用户输入2，系统检索书使用状态会话Bean getBooks（）方法和退出。
- 然后另一个JNDI名称进行查找 - "LibraryStatelessSessionBean/remote" 获得远程业务对象（状态EJB）再次列出书籍。

# 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. learn java
BUILD SUCCESSFUL (total time: 15 seconds)
```

# EJB查询语言 - EJB

EJB3.0，EJB查询语言编写自定义的查询非常方便，不用担心底层数据库的详细信息。这是很相似的HQL(Hibernate查询语言)，通常被称为按名称EJBQL。

要理解熟悉ejb的EJBQL，要做好以下几项工作。

- 步骤 1. 在数据库中创建表.
- 步骤 2. 创建一个无状态EJB.
- 步骤 3. 更新无状态的EJB。添加添加记录并获得通过实体管理器从数据库中记录的方法。
- 步骤 4. 一个基于控制台应用程序客户端将访问无状态EJB的持久化数据库中的数据。

## 创建数据库表

创建表 **books** 在默认数据库 **postgres**.

```
CREATE TABLE books (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
```

## 创建模型类

```
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }
   ...
}
```

## 创建无状态EJB

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBean

   public void addBook(Book book) {
     //persist book using entity manager
   }

   public List<Book> getBooks() {
     //get books using entity manager
   }
   ...
}
```

构建EJB模块后，我们需要一个无状态的bean，我们将在下一节要创建客户端来访问。

## 示例应用程序

让我们创建一个测试EJB应用程序来测试EJB的数据库访问机制。

| Step | Description |
| --- | --- |
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand ejb data access concepts. |
| 2 | Create *Book.java* under package *com.tutorialspoint.entity* and modify it as shown below. |
| 3 | Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as explained in the *EJB - Create Application* chapter and modify them as shown below. |
| 4 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 5 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |
| 6 | Now create the ejb client, a console based application in the same way as explained in the*EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below. |

## EJBComponent (EJB Module)

# Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;

public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

# LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

## LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBear

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EntityEjbPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      //create an ejbql expression
      String ejbQL = "From Book b where b.name like ?1";
      //create query
      Query query = entityManager.createQuery(ejbQL);
      //substitute parameter.
      query.setParameter(1, "%test%");
      //execute the query
      return query.getResultList();
   }
}
```

- 只要在JBoss部署 EjbComponent项目，会注意到jboss的日志。
- JBoss已经自动为我们的会话bean创建一个JNDI条目 - **LibraryPersistentBean/remote**.
- 我们将使用这个查询字符串来获得远程类型的业务对象 - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

## JBoss应用服务器的日志输出

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following
   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.stateless.Librar
16:30:02,723 INFO  [SessionSpecContainer] Starting jboss.j2ee:jar=E
16:30:02,723 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following

   LibraryPersistentBean/remote - EJB3.x Default Remote Business In
   LibraryPersistentBean/remote-com.tutorialspoint.stateless.Librar
...
```

# EJBTester (EJB Client)

# jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- 这些属性是用来初始化InitialContext对象的Java命名服务
- InitialContext的对象将被用于查找无状态会话bean

# EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
```

```
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEntityEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options
1\. Add Book
2\. Exit
Enter Choice: ");
   }

   private void testEntityEjb(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
         LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
```

```
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester做以下任务。

- jndi.properties中加载和初始化的InitialContext对象。
- 在testStatefulEjb（）方法，JNDI名称进行查找 - "LibraryStatelessSessionBean/remote" 以获得远程业务对象（状态EJB）。
- 然后用户显示一个库存储的用户界面和他/她被要求输入选择。
- 如果用户输入1，系统要求输入书籍名称和节省了使用无状态的会话bean addBook（）方法。会话Bean坚持这本书中通过EntityManager的调用数据库。
- 如果用户输入2，系统检索书使用状态会话Bean getBooks（）方法和退出。
- 然后另一个JNDI名称进行查找 - "LibraryStatelessSessionBean/remote"获得远程业务对象（状态EJB）再次列出书籍。

## 运行客户端访问EJB

在项目资源管理器中找到EJBTester.java。右键点击上EJBTester类，并选择**run file**.

在Netbeans控制台验证以下输出。

```
run:
*********************
Welcome to Book Store
*********************
Options
1\. Add Book
2\. Exit
Enter Choice: 1
Enter book name: Learn Testing
*********************
Welcome to Book Store
*********************
Options
1\. Add Book
2\. Exit
Enter Choice: 2
Book(s) entered so far: 1
1\. learn Testing
BUILD SUCCESSFUL (total time: 15 seconds)
```

# EJB Web Services - EJB

EJB3.0暴露会话EJB作为Web服务提供选项。@WebService注释是用来标记一类作为一个Web服务端点，并使用@WebMethod是用来作为客户端的Web方法的公开方法。

```
@Stateless
@WebService(serviceName="LibraryService")
public class LibraryPersistentBean implements LibraryPersistentBear

    ...
    @WebMethod(operationName="getBooks")
    public List<Book> getBooks()  {
       return entityManager.createQuery("From Books").getResultList(
    }
    ...
}
```

## 示例应用程序

让我们创建一个测试EJB在EJB3.0应用程序来测试BLOB/CLOB支持。

| Step | Description |
| --- | --- |
| 1 | Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Please use the project created in *EJB - Persistence* chapter as such for this chapter to understand clob/blob objects in ejb concepts. |
| 2 | Create *LibraryPersistentBean.java* under package *com.tutorialspoint.stateless*. Use *EJB - Persistence* chapter as reference. Keep rest of the files unchanged. |
| 3 | Clean and Build the application to make sure business logic is working as per the requirements. |
| 4 | Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet. |

# LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
@WebService(serviceName="LibraryService")
public class LibraryPersistentBean implements LibraryPersistentBean

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   @WebMethod(operationName="getBooks")
   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList(
   }
}
```

JBoss应用服务器的日志输出

```
10:51:37,271 INFO  [EJBContainer] STARTED EJB: com.tutorialspoint.s
10:51:37,287 INFO  [JndiSessionRegistrarBase] Binding the following

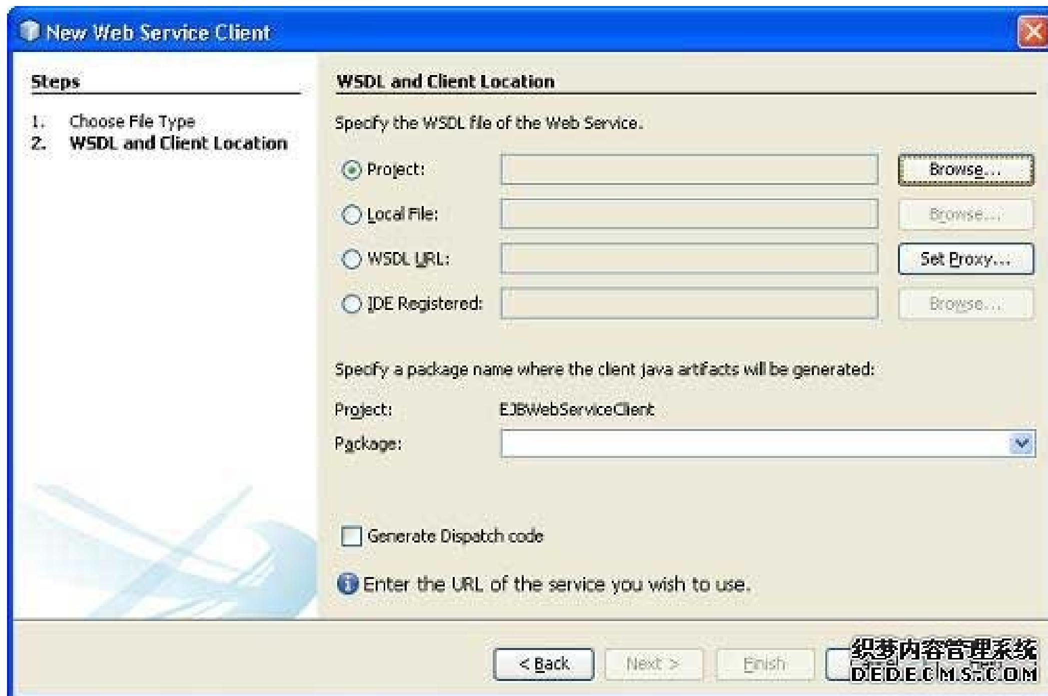
    LibraryPersistentBean/remote - EJB3.x Default Remote Business I
    LibraryPersistentBean/remote-com.tutorialspoint.stateless.Libra

10:51:37,349 INFO  [EJBContainer] STARTED EJB: com.tuturialspoint.m
10:51:37,443 INFO  [DefaultEndpointRegistry] register: jboss.ws:con
10:51:38,191 INFO  [WSDLFilePublisher] WSDL published to: file:/D:/
LibraryService3853081455302946642.wsdl
```

# 创建客户端访问EJB作为Web服务

在NetBeansIDE中，选择**File>NewProject>**类别下选择项目类型，Java项目类型为Java应用。点击Next>按钮。输入项目的名称和位置。单击“**Next>**”按钮。我们选择名为EJBWebServiceClient。

右键点击项目名称在项目exporer窗口中。选择 **New > WebService Client** .




添加EJB组件项目的LibraryPersistentBean早下创建WSDL和客户端位置使用在"编译"选项卡中添加项目“按钮。




单击“完成”按钮。在项目资源管理器验证以下结构。

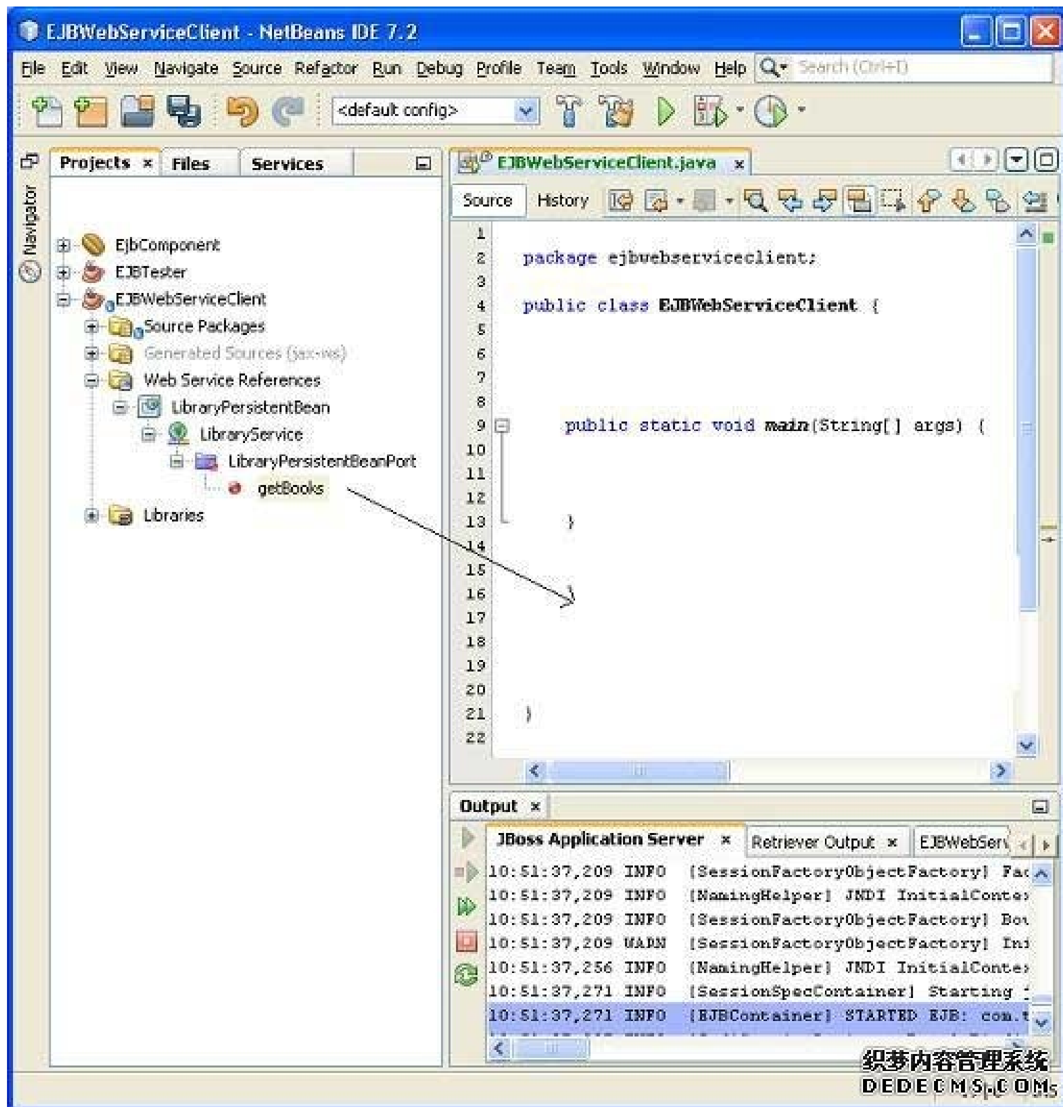



Create EJBWebServiceClient.java

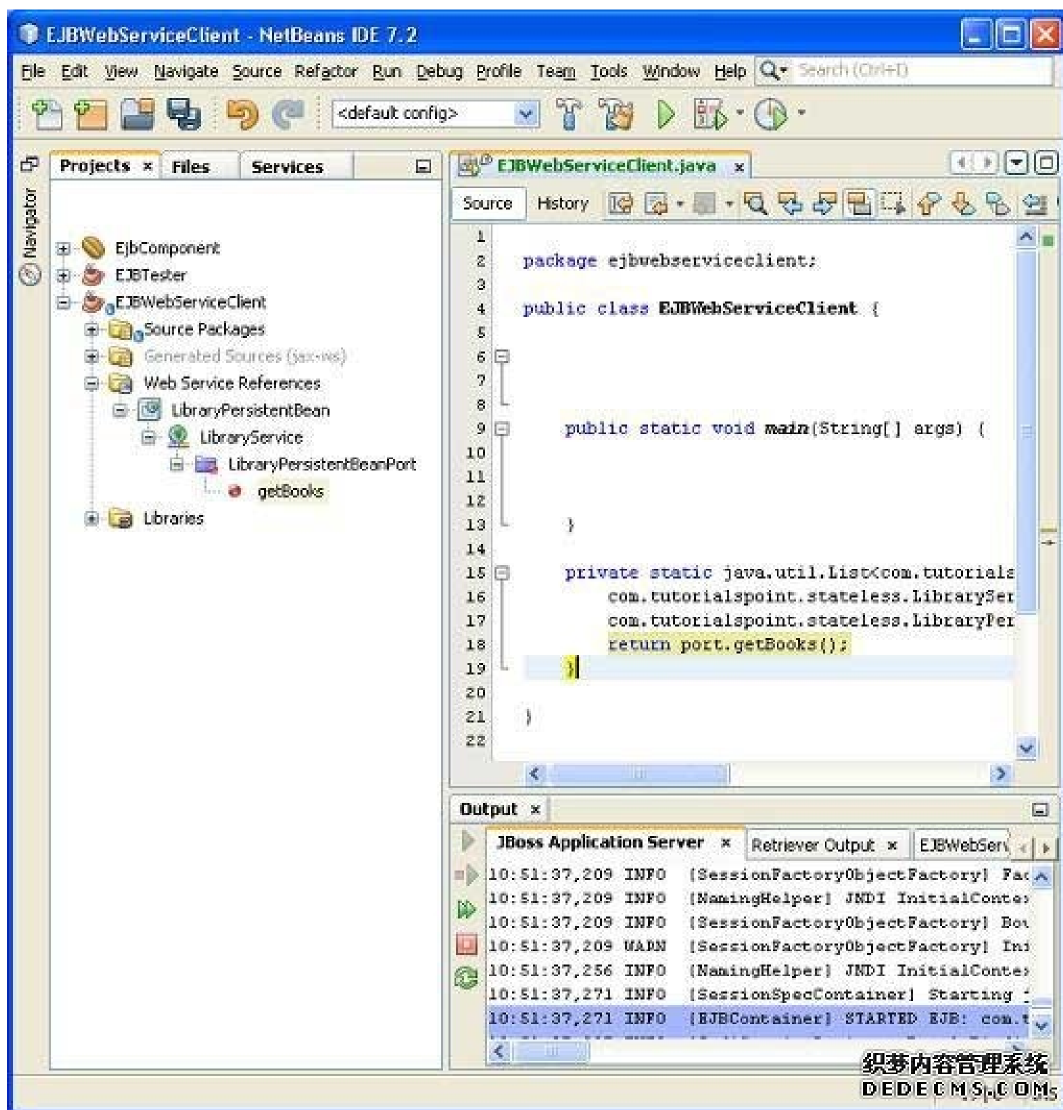
```
package ejbwebserviceclient;

public class EJBWebServiceClient {
   public static void main(String[] args) {
   }
}
```

如下图所示，选择Web服务getBooks Web方法，将其拖动到代码窗口EJBWebServiceClient。




会看到类似的输出如下所示。




更新EJBWebServiceClient的代码使用此方法。

```
package ejbwebserviceclient;

public class EJBWebServiceClient {

   public static void main(String[] args) {
      for(com.tutorialspoint.stateless.Book book:getBooks()){
         System.out.println(book.getName());
      }
   }

   private static java.util.List
   <com.tutorialspoint.stateless.Book> getBooks() {
      com.tutorialspoint.stateless.LibraryService service =
         new com.tutorialspoint.stateless.LibraryService();
      com.tutorialspoint.stateless.LibraryPersistentBean port =
         service.getLibraryPersistentBeanPort();
      return port.getBooks();
   }
}
```

## 运行客户端

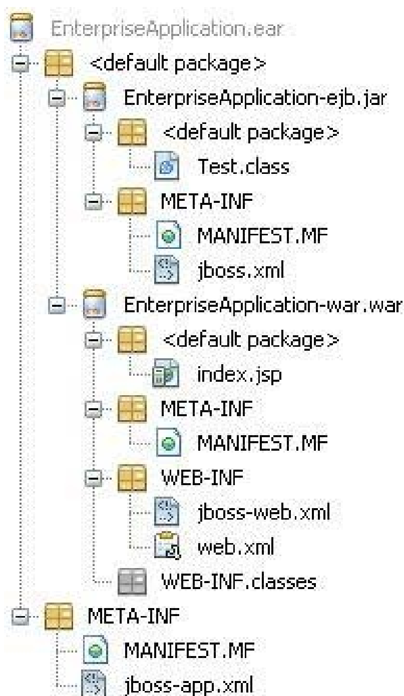
右键点击项目名称，项目资源管理器窗口。选择“**Run**”。 NetBeans将生成客户端并运行它。验证下面的输出。

```
ant -f D:\SVN\EJBWebServiceClient run
init:
Deleting: D:SVNEJBWebServiceClientuilduilt-jar.properties
deps-jar:
Updating property file: D:SVNEJBWebServiceClientuilduilt-jar.proper
wsimport-init:
wsimport-client-LibraryPersistentBean:
files are up to date
classLoader = java.net.URLClassLoader@4ce46c
SharedSecrets.getJavaNetAccess()=java.net.URLClassLoader$7@182cdac
wsimport-client-generate:
Compiling 1 source file to D:SVNEJBWebServiceClientuildclasses
compile:
run:
learn java
Learn Spring
learn JSF
Learn HTML
Learn JBoss
Learn EJB
Learn Hibernate
Learn IBatis
Times Now
learn html5
Learn images
Learn Testing
Forbes
test1
BUILD SUCCESSFUL (total time: 1 second)
```

# EJB封装应用 - EJB

封装要求应用程序使用EJB 3.0是类似于J2EE平台。 EJB组件打包成jar文件的模块被打包成ear文件的应用企业归档。 主要的企业应用程序有三个组成部分。

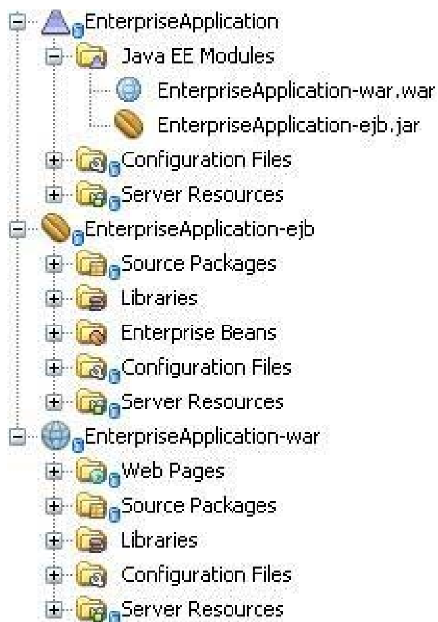
- **jar** - Java应用程序归档，包含EJB模块，EJB客户端模块和实用模块。
- **war** - Web应用程序归档，包含Web模块。
- **ear** - 企业应用程序归档包括 jars 和 war 模块.




在NetBeans它是很容易的创建，开发，打包和部署J2EE应用程序。

在NetBeans IDE中，选择**File>New Project>**选择项目类型为企业应用程序的**JavaEE**项目类型。 点击Next>按钮。输入项目的名称和位置。单击“**Finish >**“按钮。我们选择名为EnterpriseApplicaton。

选择服务器和设置。保持创建EJB模块“和”创建Web应用程序模块检查提供的默认名称。单击“完成”按钮。 NetBeans将创建下列结构在项目窗口。




项目企业在项目资源管理器中的应用上点击右键并选择“生成”。

```
ant -f D:\SVN\EnterpriseApplication dist
pre-init:
init-private:
init-userdir:
init-user:
init-project:
do-init:
post-init:
init-check:
init:
deps-jar:
deps-j2ee-archive:
EnterpriseApplication-ejb.init:
EnterpriseApplication-ejb.deps-jar:
EnterpriseApplication-ejb.compile:
EnterpriseApplication-ejb.library-inclusion-in-manifest:

Building jar: D:SVNEnterpriseApplicationEnterpriseApplication-ejbd

EnterpriseApplication-ejb.dist-ear:
EnterpriseApplication-war.init:
EnterpriseApplication-war.deps-module-jar:
EnterpriseApplication-war.deps-ear-jar:
EnterpriseApplication-ejb.init:
EnterpriseApplication-ejb.deps-jar:
EnterpriseApplication-ejb.compile:
EnterpriseApplication-ejb.library-inclusion-in-manifest:
EnterpriseApplication-ejb.dist-ear:
EnterpriseApplication-war.deps-jar:
EnterpriseApplication-war.library-inclusion-in-archive:
EnterpriseApplication-war.library-inclusion-in-manifest:
EnterpriseApplication-war.compile:
```

```
EnterpriseApplication-war.compile-jsps:
EnterpriseApplication-war.do-ear-dist:

Building jar: D:SVNEnterpriseApplicationEnterpriseApplication-wardi

EnterpriseApplication-war.dist-ear:
pre-pre-compile:
pre-compile:
Copying 1 file to D:SVNEnterpriseApplicationuild
Copying 1 file to D:SVNEnterpriseApplicationuild
do-compile:
post-compile:
compile:
pre-dist:
do-dist-without-manifest:
do-dist-with-manifest:

Building jar: D:SVNEnterpriseApplicationdistEnterpriseApplication.e

post-dist:
dist:
BUILD SUCCESSFUL (total time: 1 second)
```

在这里，你可以看到每个jar，war和ear文件带有一个META-INF文件夹，按照J2EE规范的元数据。

# Guava教程

## Guava 是什么?

Guava是一种基于开源的Java库，其中包含谷歌正在由他们很多项目使用的很多核心库。这个库是为了方便编码，并减少编码错误。这个库提供用于集合，缓存，支持原语，并发性，常见注解，字符串处理，I/O和验证的实用方法。

## Guava的好处

- 标准化 - Guava库是由谷歌托管。
- 高效 - 可靠，快速和有效的扩展JAVA标准库
- 优化 -Guava库经过高度的优化。
- 函数式编程 -增加JAVA功能和处理能力。
- 实用程序 - 提供了经常需要在应用程序开发的许多实用程序类。
- 验证 -提供标准的故障安全验证机制。
- 最佳实践 - 强调最佳的做法。

考虑下面的代码片段。

```
public class GuavaTester {
   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();
      Integer a =  null;
      Integer b =  new Integer(10);
      System.out.println(guavaTester.sum(a,b));
   }

   public Integer sum(Integer a, Integer b){
      return a + b;
   }
}
```

运行程序，看到如下结果。

```
Exception in thread "main" java.lang.NullPointerException
    at GuavaTester.sum(GuavaTester.java:13)
    at GuavaTester.main(GuavaTester.java:9)
```

以下是该代码的问题。

- sum() 不采取任何的保护传递的参数为null。
- 调用函数也并不担心传递一个null 到sum（）方法而产生意外。
- 当程序运行时，NullPointerException异常发生。
- 为了避免上述问题，null检查要在每个这样存在问题地方。

让我们来看看使用Optional，Guava 提供实用工具类来标准化方式解决上述问题。

```
import com.google.common.base.Optional;

public class GuavaTester {
   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();

      Integer invalidInput = null;
      Optional<Integer> a =  Optional.of(invalidInput);
      Optional<Integer> b =  Optional.of(new Integer(10));
      System.out.println(guavaTester.sum(a,b));
   }

   public Integer sum(Optional<Integer> a, Optional<Integer> b){
      return a.get() + b.get();
   }
}
```

运行程序，看到结果如下。

```
Exception in thread "main" java.lang.NullPointerException
    at com.google.common.base.Preconditions.checkNotNull(Precondit
    at com.google.common.base.Optional.of(Optional.java:85)
    at GuavaTester.main(GuavaTester.java:8)
```

让我们来了解上述程序的一些重要概念。

- **Optional** - 实用类，使代码使用null能够正常。
- **Optional.of** - 返回要用作参数Optional类的实例。检查传递的值是否为null。
- **Optional.get** - 获取输入存储在Optional 类的值。

使用Optional类，可以方便地查看调用者方法来传递参数正确与否。

# Guava环境设置 - Guava教程

## Guava本地环境设置

这部分指导如何下载和设置Java在机器上。请按照以下步骤来设置环境。

Java SE免费提供链接：下载Java。所以，根据操作系统下载对应版本。

按照说明下载java和运行.exe 在机器上，并按说明安装Java。一旦机器上安装了Java，还需要设置环境变量指向正确的JAVA安装目录：

## Windows 2000/XP设置的路径：

假设你已经安装JAVA在 c:Program Filesjavajdk目录下：

- 在“我的电脑”右键单击并选择“属性”。
- 在“高级”选项卡下单击“环境变量”按钮。
- 现在，改变“Path”变量，因此，它也包含了路径Java可执行文件。例如，路径当前默认设置为'C:WINDOWSSYSTEM32',然后改变路径修改后就是这样：'C:WINDOWSSYSTEM32;c:Program Filesjavajdkin'.

## Windows95/98/ME设置路径：

假设你已经安装JAVA在 c:Program Filesjavajdk目录下：

- 编辑文件 'C:autoexec.bat' 并添加以下行到文件的结尾： 'SET PATH=%PATH%;C:Program Filesjavajdkin'

## Linux， UNIX， Solaris和FreeBSD设置路径：

环境变量PATH应设置为指向已安装的Java二进制文件的位置。请参考您的shell文件，如果有麻烦可以这样做。

例如，如果使用的是bash shell，那么将下面的行添加到末尾'.bashrc: export PATH=/path/to/java:$PATH'

## 流行的Java编辑器：

写Java程序，需要一个文本编辑器。在市场上有很多可用的更复杂的IDE。可以考虑下列之一：

- Notepad: 在Windows机器上，可以使用像记事本的简单的文本编辑器（本教程推荐），TextPad。
- Netbeans:是一个Java IDE，它是开源和免费的，可从以下地址下载 http://www.netbeans.org/index.html.
- Eclipse: 也是一个Java IDE由Eclipse开源社区开发，可以下载 http://www.eclipse.org/.

## 下载Guava存档

从guava-18.0.jar下载Guava jar文件的最新版本。在写这篇教程的时候，下载最新的版本是 guava-18.0.jar 并拷贝到 C:>Guava 目录。

| OS | 归档名称 |
|---|---|
| Windows | guava-18.0.jar |
| Linux | guava-18.0.jar |
| Mac | guava-18.0.jar |

## 设置Guava环境

设置Guava_HOME环境变量指向Guava jar 存储在计算机上的目录位置。假设，我们已经提取guava-18.0.jar在各种操作系统的Guava文件夹，如下所示。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 Guava_HOME 到 C:Guava |
| Linux | export Guava_HOME=/usr/local/Guava |
| Mac | export Guava_HOME=/Library/Guava |

## 设置CLASSPATH变量

设置CLASSPATH环境变量指向Guava jar位置。假设，我们已经存储guava-18.0.jar的Guava文件夹在不同的操作系统如下。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 CLASSPATH 到 %CLASSPATH%;%Guava_HOME%guava-18.0.jar;.; |
| Linux | export CLASSPATH=$CLASSPATH:$Guava_HOME/guava-18.0.jar:. |
| Mac | export CLASSPATH=$CLASSPATH:$Guava_HOME/guava-18.0.jar:. |

# Guava Optional类 - Guava教程

Optional用于包含非空对象的不可变对象。 Optional对象，用于不存在值表示null。这个类有各种实用的方法，以方便代码来处理为可用或不可用，而不是检查null值。

## 类声明

以下是com.google.common.base.Optional<T>类的声明：

```
@GwtCompatible(serializable=true)
public abstract class Optional<T>
   extends Object
      implements Serializable
```

## 类方法

| S.N. | 方法及说明 |
| --- | --- |
| 1 | **static <T> Optional<T> absent()** 返回没有包含的参考Optional的实例。 |
| 2 | **abstract Set<T> asSet()** 返回一个不可变的单集的唯一元素所包含的实例(如果存在);否则为一个空的不可变的集合。 |
| 3 | **abstract boolean equals(Object object)** 返回true如果对象是一个Optional实例，无论是包含引用彼此相等或两者都不存在。 |
| 4 | **static <T> Optional<T> fromNullable(T nullableReference)** 如果nullableReference非空，返回一个包含引用Optional实例;否则返回absent()。 |
| 5 | **abstract T get()** 返回所包含的实例，它必须存在。 |
| 6 | **abstract int hashCode()** 返回此实例的哈希码。 |
| 7 | **abstract boolean isPresent()** 返回true，如果这支架包含一个(非空)的实例。 |
| 8 | **static <T> Optional<T> of(T reference)** 返回包含给定的非空引用Optional实例。 |
| 9 | **abstract Optional<T> or(Optional<? extends T> secondChoice)** 返回此Optional，如果它有一个值存在; 否则返回secondChoice。 |
| 10 | **abstract T or(Supplier<? extends T> supplier)** 返回所包含的实例(如果存在); 否则supplier.get()。 |
| 11 | **abstract T or(T defaultValue)** 返回所包含的实例(如果存在);否则为默认值。 |
| 12 | **abstract T orNull()** 返回所包含的实例(如果存在);否则返回null。 |
| 13 | **static <T> Iterable<T> presentInstances(Iterable<? extends Optional<? extends T>> optionals)** 从提供的optionals返回每个实例的存在的值，从而跳过absent()。 |
| 14 | **abstract String toString()** 返回此实例的字符串表示。 |
| 15 | **abstract <V> Optional<V> transform(Function<? super T,V> function)** 如果实例存在，则它被转换给定的功能;否则absent()被返回。 |

## 继承的方法

这个类继承了以下类的方法：

- java.lang.Object

# Optional示例：

使用所选择的编辑器，创建下面的java程序，比如 C:/> Guava

GuavaTester.java

```
import com.google.common.base.Optional;

public class GuavaTester {
   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();

      Integer value1 =  null;
      Integer value2 =  new Integer(10);
      //Optional.fromNullable - allows passed parameter to be null
      Optional<Integer> a = Optional.fromNullable(value1);
      //Optional.of - throws NullPointerException if passed paramet
      Optional<Integer> b = Optional.of(value2);

      System.out.println(guavaTester.sum(a,b));
   }

   public Integer sum(Optional<Integer> a, Optional<Integer> b){
      //Optional.isPresent - checks the value is present or not
      System.out.println("First parameter is present: " + a.isPrese

      System.out.println("Second parameter is present: " + b.isPres

      //Optional.or - returns the value if present otherwise returr
      //the default value passed.
      Integer value1 = a.or(new Integer(0));

      //Optional.get - gets the value, value should be present
      Integer value2 = b.get();

      return value1 + value2;
   }
}
```

## 验证结果

使用javac编译器编译如下类

```
C:\Guava>javac GuavaTester.java
```

现在运行GuavaTester看到的结果

```
C:\Guava>java GuavaTester
```

看到结果。

```
First parameter is present: false
Second parameter is present: true
10
```

# Guava Preconditions类 - Guava教程

Preconditions提供静态方法来检查方法或构造函数，被调用是否给定适当的参数。它检查的先决条件。其方法失败抛出IllegalArgumentException。

## 类声明

以下是com.google.common.base.Preconditions类的声明：

```
@GwtCompatible
public final class Preconditions
   extends Object
```

## 类方法

| S.N. | 方法及说明 |
| --- | --- |
| 1 | **static void checkArgument(boolean expression)** 确保涉及的一个或多个参数来调用方法表达式的真相。 |
| 2 | **static void checkArgument(boolean expression, Object errorMessage)** 确保涉及的一个或多个参数来调用方法表达式的真相。 |
| 3 | **static void checkArgument(boolean expression, String errorMessageTemplate, Object... errorMessageArgs)** 确保涉及的一个或多个参数来调用方法表达式的真相。 |
| 4 | **static int checkElementIndex(int index, int size)** 确保索引指定一个数组，列表或尺寸大小的字符串有效的元素。 |
| 5 | **static int checkElementIndex(int index, int size, String desc)** 确保索引指定一个数组，列表或尺寸大小的字符串有效的元素。 |
| 6 | **static <T> T checkNotNull(T reference)** 确保对象引用作为参数传递给调用方法不为空。 |
| 7 | **static <T> T checkNotNull(T reference, Object errorMessage)** 确保对象引用作为参数传递给调用方法不为空。 |
| 8 | **static <T> T checkNotNull(T reference, String errorMessageTemplate, Object... errorMessageArgs)** 确保对象引用作为参数传递给调用方法不为空。 |
| 9 | **static int checkPositionIndex(int index, int size)** 确保索引指定一个数组，列表或尺寸大小的字符串的有效位置。 |
| 10 | **static int checkPositionIndex(int index, int size, String desc)** 确保索引指定一个数组，列表或尺寸大小的字符串的有效位置。 |
| 11 | **static void checkPositionIndexes(int start, int end, int size)** 确保开始和结束指定数组，列表或字符串大小有效的位置，并按照顺序。 |
| 12 | **static void checkState(boolean expression)** 确保涉及调用实例的状态，但不涉及任何参数来调用方法表达式的真相。 |
| 13 | **static void checkState(boolean expression, Object errorMessage)** 确保涉及调用实例的状态，但不涉及任何参数来调用方法表达式的真相。 |
| 14 | **static void checkState(boolean expression, String errorMessageTemplate, Object... errorMessageArgs)** 确保涉及调用实例的状态，但不涉及任何参数来调用方法表达式的真相。 |

## 继承的方法

这个类继承了以下类方法：

- java.lang.Object

## Preconditions 示例

使用所选择的编辑器，创建下面的java程序比如 C:/> Guava

*GuavaTester.java*

```
import com.google.common.base.Preconditions;

public class GuavaTester {

   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();
      try {
         System.out.println(guavaTester.sqrt(-3.0));
      }catch(IllegalArgumentException e){
         System.out.println(e.getMessage());
      }
      try {
         System.out.println(guavaTester.sum(null,3));
      }catch(NullPointerException e){
         System.out.println(e.getMessage());
      }
      try {
         System.out.println(guavaTester.getValue(6));
      }catch(IndexOutOfBoundsException e){
         System.out.println(e.getMessage());
      }
   }

   public double sqrt(double input) throws IllegalArgumentExceptio
      Preconditions.checkArgument(input > 0.0,
         "Illegal Argument passed: Negative value %s.", input);
      return Math.sqrt(input);
   }

   public int sum(Integer a, Integer b){
      a = Preconditions.checkNotNull(a,
         "Illegal Argument passed: First parameter is Null.");
      b = Preconditions.checkNotNull(b,
         "Illegal Argument passed: Second parameter is Null.");
      return a+b;
   }

   public int getValue(int input){
      int[] data = {1,2,3,4,5};
      Preconditions.checkElementIndex(input,data.length,
         "Illegal Argument passed: Invalid index.");
      return 0;
   }
}
```

## 验证结果

使用javac编译器编译如下类

```
C:\Guava>javac GuavaTester.java
```

现在运行GuavaTester看到的结果

```
C:\Guava>java GuavaTester
```

看到结果。

```
Illegal Argument passed: Negative value -3.0.
Illegal Argument passed: First parameter is Null.
Illegal Argument passed: Invalid index. (6) must be less than size
```

# Guava Ordering类 - Guava教程

Ordering(排序)可以被看作是一个丰富的比较具有增强功能的链接，多个实用方法，多类型排序功能等。

## 类声明

以下是com.google.common.collect.Ordering<T>类的声明：

```
@GwtCompatible
public abstract class Ordering<T>
   extends Object
      implements Comparator<T>
```

## 类方法

| S.N. | 方法及说明 |
|---|---|
| 1 | **static Ordering<Object> allEqual()** 返回一个排序，它把所有的值相等，说明“没有顺序。”通过这个顺序以任何稳定的排序算法的结果，在改变没有顺序元素。 |
| 2 | **static Ordering<Object> arbitrary()** 返回一个任意顺序对所有对象，其中compare(a, b) == 0 意味着a == b（身份平等）。 |
| 3 | **int binarySearch(List<? extends T> sortedList, T key)** 搜索排序列表使用键的二进制搜索算法。 |
| 4 | **abstract int compare(T left, T right)** 比较两个参数的顺序。 |
| 5 | **<U extends T> Ordering<U> compound(Comparator<? super U> secondaryComparator)** 返回首先使用排序这一点，但它排序中的“tie”，然后委托给secondaryComparator事件。 |
| 6 | **static <T> Ordering<T> compound(Iterable<? extends Comparator<? super T>> comparators)** 返回一个排序它尝试每个给定的比较器，以便直到一个非零结果找到，返回该结果，并返回零仅当所有比较器返回零。 |
| 7 | **static <T> Ordering<T> explicit(List<T> valuesInOrder)** 返回根据它们出现的定列表中的顺序比较对象进行排序。 |
| 8 | **static <T> Ordering<T> explicit(T leastValue, T... remainingValuesInOrder)** 返回根据它们所赋予本方法的顺序进行比较的对象进行排序。 |

| | |
|---|---|
| 9 | **static <T> Ordering<T> from(Comparator<T> comparator)** 返回基于现有的比较实例进行排序。 |
| 10 | **<E extends T> List<E> greatestOf(Iterable<E> iterable, int k)** 返回根据这个顺序给出迭代，为了从最大到最小的k个最大的元素。 |
| 11 | **<E extends T> List<E> greatestOf(Iterator<E> iterator, int k)** 返回从给定的迭代器按照这个顺序，从最大到最小k个最大的元素。 |
| 12 | **<E extends T> ImmutableList<E> immutableSortedCopy(Iterable<E> elements)** 返回包含的元素排序这种排序的不可变列表。 |
| 13 | **boolean isOrdered(Iterable<? extends T> iterable)** 返回true如果在迭代后的第一个的每个元素是大于或等于在它之前，根据该排序的元素。 |
| 14 | **boolean isStrictlyOrdered(Iterable<? extends T> iterable)** 返回true如果在迭代后的第一个的每个元素是严格比在它之前，根据该排序的元素更大。 |
| 15 | **<E extends T> List<E> leastOf(Iterable<E> iterable, int k)** 返回根据这个顺序给出迭代，从而从低到最大的k个最低的元素。 |
| 16 | **<E extends T> List<E> leastOf(Iterator<E> elements, int k)** 返回第k从给定的迭代器，按照这个顺序从最低到最大至少元素。 |
| 17 | **<S extends T> Ordering<Iterable<S>> lexicographical()** 返回一个新的排序它通过比较对应元素两两直到非零结果发现排序迭代;规定“字典顺序”。 |
| 18 | **<E extends T> E max(E a, E b)** 返回两个值按照这个顺序的较大值。 |
| 19 | **<E extends T> E max(E a, E b, E c, E... rest)** 返回指定的值，根据这个顺序是最大的。 |
| 20 | **<E extends T> E max(Iterable<E> iterable)** 返回指定的值，根据这个顺序是最大的。 |
| 21 | **<E extends T> E max(Iterator<E> iterator)** 返回指定的值，根据这个顺序是最大的。 |
| 22 | **<E extends T> E min(E a, E b)** 返回两个值按照这个顺序的较小者。 |
| 23 | **<E extends T> E min(E a, E b, E c, E... rest)** 返回最少指定的值，根据这个顺序。 |
| 24 | **<E extends T> E min(Iterable<E> iterable)** 返回最少指定的值，根据这个顺序。 |
| 25 | **<E extends T> E min(Iterator<E> iterator)** 返回最少指定的值，根据这个顺序。 |
| 26 | **static <C extends Comparable> Ordering<C> natural()** 返回使用值 |

| | |
|---|---|
| 26 | 的自然顺序排序序列化。 |
| 27 | **<S extends T> Ordering<S> nullsFirst()** 返回对待null小于所有其他值，并使用此来比较非空值排序。 |
| 28 | **<S extends T> Ordering<S> nullsLast()** 返回对待null作为大于所有其他值，并使用这个顺序来比较非空值排序。 |
| 29 | **<F> Ordering<F> onResultOf(Function<F,? extends T> function)** 返回一个新的排序在F上，首先应用功能给它们，然后比较使用此这些结果的顺序元素。 |
| 30 | **<S extends T> Ordering<S> reverse()** 返回相反顺序; 顺序相当于Collections.reverseOrder（Comparator）。 |
| 31 | **<E extends T> List<E> sortedCopy(Iterable<E> elements)** 返回包含的元素排序此排序可变列表;使用这个只有在结果列表可能需要进一步修改，或可能包含null。 |
| 32 | **static Ordering<Object> usingToString()** 返回由它们的字符串表示的自然顺序，toString()比较对象进行排序。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# Ordering 示例

使用所选择的编辑器，创建下面的java程序比如 C:/> Guava

GuavaTester.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import com.google.common.collect.Ordering;

public class GuavaTester {
   public static void main(String args[]){
      List<Integer> numbers = new ArrayList<Integer>();
      numbers.add(new Integer(5));
      numbers.add(new Integer(2));
      numbers.add(new Integer(15));
      numbers.add(new Integer(51));
      numbers.add(new Integer(53));
      numbers.add(new Integer(35));
```

```
        numbers.add(new Integer(32));
        numbers.add(new Integer(43));
        numbers.add(new Integer(16));

        Ordering ordering = Ordering.natural();
        System.out.println("Input List: ");
        System.out.println(numbers);

        Collections.sort(numbers,ordering );
        System.out.println("Sorted List: ");
        System.out.println(numbers);

        System.out.println("======================");
        System.out.println("List is sorted: " + ordering.isOrdered(nu
        System.out.println("Minimum: " + ordering.min(numbers));
        System.out.println("Maximum: " + ordering.max(numbers));

        Collections.sort(numbers,ordering.reverse());
        System.out.println("Reverse: " + numbers);

        numbers.add(null);
        System.out.println("Null added to Sorted List: ");
        System.out.println(numbers);

        Collections.sort(numbers,ordering.nullsFirst());
        System.out.println("Null first Sorted List: ");
        System.out.println(numbers);
        System.out.println("======================");

        List<String> names = new ArrayList<String>();
        names.add("Ram");
        names.add("Shyam");
        names.add("Mohan");
        names.add("Sohan");
        names.add("Ramesh");
        names.add("Suresh");
        names.add("Naresh");
        names.add("Mahesh");
        names.add(null);
        names.add("Vikas");
        names.add("Deepak");

        System.out.println("Another List: ");
        System.out.println(names);

        Collections.sort(names,ordering.nullsFirst().reverse());
        System.out.println("Null first then reverse sorted list: ");
        System.out.println(names);
    }
}
```

# 验证结果

使用javac编译器编译如下类

```
C:\Guava>javac GuavaTester.java
```

现在运行GuavaTester看到的结果

```
C:\Guava>java GuavaTester
```

看到结果。

```
Input List:
[5, 2, 15, 51, 53, 35, 45, 32, 43, 16]
Sorted List:
[2, 5, 15, 16, 32, 35, 43, 45, 51, 53]
======================
List is sorted: true
Minimum: 2
Maximum: 53
Reverse: [53, 51, 45, 43, 35, 32, 16, 15, 5, 2]
Null added to Sorted List:
[53, 51, 45, 43, 35, 32, 16, 15, 5, 2, null]
Null first Sorted List:
[null, 2, 5, 15, 16, 32, 35, 43, 45, 51, 53]
======================
Another List:
[Ram, Shyam, Mohan, Sohan, Ramesh, Suresh, Naresh, Mahesh, null, Vi
Null first then reverse sorted list:
[Vikas, Suresh, Sohan, Shyam, Ramesh, Ram, Naresh, Mohan, Mahesh, [
```

# Guava Objects类 - Guava教程

Objects类提供适用于所有对象，如equals, hashCode等辅助函数

## 类声明

以下是com.google.common.base.Objects类的声明：

```
@GwtCompatible
public final class Objects
   extends Object
```

## 类方法

| S.N. | 方法及说明 |
|---|---|
| 1 | **static boolean equal(Object a, Object b)** 确定两个可能是空的对象是否相等。 |
| 2 | **static <T> T firstNonNull(T first, T second)** 不推荐使用。使用MoreObjects.firstNonNull（T，T）来代替。定于2016年6月去除该方法。 |
| 3 | **static int hashCode(Object... objects)** 生成多个值的哈希码。 |
| 4 | **static Objects.ToStringHelper toStringHelper(Class<?> clazz)** 不推荐使用。使用MoreObjects.toStringHelper（Class）来代替。定于2016年6月去除该方法。 |
| 5 | **static Objects.ToStringHelper toStringHelper(Object self)** 不推荐使用。使用MoreObjects.toStringHelper（Object）来代替。定于2016年6月去除该方法。 |
| 6 | **static Objects.ToStringHelper toStringHelper(String className)** 不推荐使用。使用MoreObjects.toStringHelper（String）来代替。定于2016年6月去除该方法。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# Objects 示例

使用所选择的编辑器，创建下面的java程序比如 **C:/> Guava**

*GuavaTester.java*

```
import com.google.common.base.Objects;

public class GuavaTester {
   public static void main(String args[]){
      Student s1 = new Student("Mahesh", "Parashar", 1, "VI");
      Student s2 = new Student("Suresh", null, 3, null);

      System.out.println(s1.equals(s2));
      System.out.println(s1.hashCode());
      System.out.println(
      Objects.toStringHelper(s1)
         .add("Name",s1.getFirstName()+" " + s1.getLastName())
         .add("Class", s1.getClassName())
         .add("Roll No", s1.getRollNo())
         .toString());
   }
}

class Student {
   private String firstName;
   private String lastName;
   private int rollNo;
   private String className;

   public Student(String firstName, String lastName, int rollNo, St
      this.firstName = firstName;
      this.lastName = lastName;
      this.rollNo = rollNo;
      this.className = className;
   }

   @Override
   public boolean equals(Object object){
      if(!(object instanceof Student) || object == null){
         return false;
      }
      Student student = (Student)object;
      // no need to handle null here
      // Objects.equal("test", "test") == true
      // Objects.equal("test", null) == false
      // Objects.equal(null, "test") == false
      // Objects.equal(null, null) == true
      return Objects.equal(firstName, student.firstName) // first n
         && Objects.equal(lastName, student.lastName) // last name
         && Objects.equal(rollNo, student.rollNo)
         && Objects.equal(className, student.className);// class na
```

```
    }

    @Override
    public int hashCode(){
       //no need to compute hashCode by self
       return Objects.hashCode(className,rollNo);
    }
    public String getFirstName() {
       return firstName;
    }
    public void setFirstName(String firstName) {
       this.firstName = firstName;
    }
    public String getLastName() {
       return lastName;
    }
    public void setLastName(String lastName) {
       this.lastName = lastName;
    }
    public int getRollNo() {
       return rollNo;
    }
    public void setRollNo(int rollNo) {
       this.rollNo = rollNo;
    }
    public String getClassName() {
       return className;
    }
    public void setClassName(String className) {
       this.className = className;
    }
 }
```

## 验证结果

使用javac编译器编译如下类

```
C:\Guava>javac GuavaTester.java
```

现在运行GuavaTester看到的结果

```
C:\Guava>java GuavaTester
```

看到结果。

```
false
85871
Student{Name=Mahesh Parashar, Class=VI, Roll No=1}
```

# Guava Range类 - Guava教程

Range 表示一个间隔或一个序列。它被用于获取一组数字/串在一个特定范围之内。

## 类声明

以下是com.google.common.collect.Range<C>类的声明：

```
@GwtCompatible
public final class Range<C extends Comparable>
   extends Object
      implements Predicate<C>, Serializable
```

## 方法

| S.N. | 方法及说明 |
|---|---|
| 1 | **static <C extends Comparable<?>> Range<C> all()** 返回包含C型的每一个值范围 |
| 2 | **boolean apply(C input)Deprecated.** 只有提供满足谓词接口;使用包含(C)来代替。 |
| 3 | **static <C extends Comparable<?>> Range<C> atLeast(C endpoint)** 返回包含大于或等于终点(endpoint)的所有值的范围内。 |
| 4 | **static <C extends Comparable<?>> Range<C> atMost(C endpoint)** 返回包含的所有值小于或等于终点的范围内。 |
| 5 | **Range<C> canonical(DiscreteDomain<C> domain)** 返回此范围内，在给定域中的规范形式。 |
| 6 | **static <C extends Comparable<?>> Range<C> closed(C lower, C upper)** 返回包含大于所有值或等于降低且小于或等于上限的范围内。 |
| 7 | **static <C extends Comparable<?>> Range<C> closedOpen(C lower, C upper)** 返回包含大于或等于下限和所有值严格大于上限以下的范围内。 |
| 8 | **boolean contains(C value)** 返回true，如果值是这个范围的范围之内。 |
| 9 | **boolean containsAll(Iterable<? extends C> values)** 如果值每一个元素都包含在这个范围内，则返回 true。 |

| | |
|---|---|
| 10 | **static <C extends Comparable<?>> Range<C> downTo(C endpoint, BoundType boundType)** 返回的范围内的给定的端点，它可以是包容性（闭合）或专用（开），没有上限。 |
| 11 | **static <C extends Comparable<?>> Range<C> encloseAll(Iterable<C> values)** 返回包含所有给定值的最小范围内。 |
| 12 | **boolean encloses(Range<C> other)** 返回true，如果其他的边界不在该范围的边界之外延伸。 |
| 13 | **boolean equals(Object object)** 返回true，如果对象是具有相同端点和绑定类型，这个范围内的范围。 |
| 14 | **static <C extends Comparable<?>> Range<C> greaterThan(C endpoint)** 返回一个包含所有值严格大于端点的范围内。 |
| 15 | **int hashCode()** 返回此范围内的哈希码。 |
| 16 | **boolean hasLowerBound()** 如果此范围内具有更低的终点返回true。 |
| 17 | **boolean hasUpperBound()** 如果此范围内有上端点返回true。 |
| 18 | **Range<C> intersection(Range<C> connectedRange)** 返回由两者范围和connectedRange封闭，如果这样的范围存在的最大范围。 |
| 19 | **boolean isConnected(Range<C> other)** 如果存在这是由两者此范围和其他封闭（可能为空）的范围，则返回true。 |
| 20 | **boolean isEmpty()** 返回true，如果这个范围是形式 [v..v) 或 (v..v]. |
| 21 | **static <C extends Comparable<?>> Range<C> lessThan(C endpoint)** 返回一个包含所有值严格小于端点的范围内。 |
| 22 | **BoundType lowerBoundType()** 返回类型这个范围的下限：如果范围包括它的下端点BoundType.CLOSED，如果没有BoundType.OPEN。 |
| 23 | **C lowerEndpoint()** 返回该范围的较低端点。 |
| 24 | **static <C extends Comparable<?>> Range<C> open(C lower, C upper)** 返回一个包含所有值严格大于下限和严格比上端更小一个范围。 |
| 25 | **static <C extends Comparable<?>> Range<C> openClosed(C lower, C upper)** 返回包含所有值严格低于更大且小于或等于上限的范围内。 |
| 26 | **static <C extends Comparable<?>> Range<C> range(C lower, BoundType lowerType, C upper, BoundType upperType)** 返回包含任何值由下到上，每个端点可以是包容性（关闭）或专用（开）的范围。 |
| 27 | **static <C extends Comparable<?>> Range<C> singleton(C value)** 返回包含只在给定范围内的值。 |
| 28 | **Range<C> span(Range<C> other)** 返回最小的范围包围两者这个范围和other等。 |

| 29 | **String toString()** 返回该范围内的字符串表示，如“[3..5）”（其他实例列在类文档）。 |
|---|---|
| 30 | **BoundType upperBoundType()** 返回类型此范围的上限：如果范围包括其上的端点返回BoundType.CLOSED，如果没有返回BoundType.OPEN。 |
| 31 | **C upperEndpoint()** 返回此范围的上限端点。 |
| 32 | **static <C extends Comparable<?>> Range<C> upTo(C endpoint, BoundType boundType)** 返回一个范围，没有下限到给定的端点，它可以是包容性（闭合）或专用（开）。 |

# 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# Range 例子

选择使用任何编辑器创建以下java程序在 **C:/> Guava**

*GuavaTester.java*

```
import com.google.common.collect.ContiguousSet;
import com.google.common.collect.DiscreteDomain;
import com.google.common.collect.Range;
import com.google.common.primitives.Ints;

public class GuavaTester {

   public static void main(String args[]){
      GuavaTester tester = new GuavaTester();
      tester.testRange();
   }

   private void testRange(){

      //create a range [a,b] = { x | a <= x <= b}
      Range<Integer> range1 = Range.closed(0, 9);
      System.out.print("[0,9] : ");
      printRange(range1);
      System.out.println("5 is present: " + range1.contains(5));
      System.out.println("(1,2,3) is present: " + range1.containsA
      System.out.println("Lower Bound: " + range1.lowerEndpoint())
      System.out.println("Upper Bound: " + range1.upperEndpoint())

      //create a range (a,b) = { x | a < x < b}
```

```
      Range<Integer> range2 = Range.open(0, 9);
      System.out.print("(0,9) : ");
      printRange(range2);

      //create a range (a,b] = { x | a < x <= b}
      Range<Integer> range3 = Range.openClosed(0, 9);
      System.out.print("(0,9] : ");
      printRange(range3);

      //create a range [a,b) = { x | a <= x < b}
      Range<Integer> range4 = Range.closedOpen(0, 9);
      System.out.print("[0,9) : ");
      printRange(range4);

      //create an open ended range (9, infinity
      Range<Integer> range5 = Range.greaterThan(9);
      System.out.println("(9,infinity) : ");
      System.out.println("Lower Bound: " + range5.lowerEndpoint());
      System.out.println("Upper Bound present: " + range5.hasUpperE

      Range<Integer> range6 = Range.closed(3, 5);
      printRange(range6);

      //check a subrange [3,5] in [0,9]
      System.out.println("[0,9] encloses [3,5]:" + range1.encloses(

      Range<Integer> range7 = Range.closed(9, 20);
      printRange(range7);
      //check ranges to be connected
      System.out.println("[0,9] is connected [9,20]:" + range1.isCo

      Range<Integer> range8 = Range.closed(5, 15);

      //intersection
      printRange(range1.intersection(range8));

      //span
      printRange(range1.span(range8));
   }

   private void printRange(Range<Integer> range){
      System.out.print("[ ");
      for(int grade : ContiguousSet.create(range, DiscreteDomain.in
         System.out.print(grade +" ");
      }
      System.out.println("]");
   }
}
```

# 验证结果

使用javac编译器编译如下类

```
C:\Guava>javac GuavaTester.java
```

现在运行GuavaTester看到的结果

```
C:\Guava>java GuavaTester
```

看到结果。

```
[0,9] : [ 0 1 2 3 4 5 6 7 8 9 ]
5 is present: true
(1,2,3) is present: true
Lower Bound: 0
Upper Bound: 9
(0,9) : [ 1 2 3 4 5 6 7 8 ]
(0,9] : [ 1 2 3 4 5 6 7 8 9 ]
[0,9) : [ 0 1 2 3 4 5 6 7 8 ]
(9,infinity) :
Lower Bound: 9
Upper Bound present: false
[ 3 4 5 ]
[0,9] encloses [3,5]:true
[ 9 10 11 12 13 14 15 16 17 18 19 20 ]
[0,9] is connected [9,20]:true
[ 5 6 7 8 9 ]
[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ]
```

# Guava Throwables类 - Guava教程

Throwable类提供了相关的Throwable接口的实用方法。

## 类声明

以下是com.google.common.base.Throwables类的声明：

```
public final class Throwables
   extends Object
```

## 类方法

| S.N. | 方法及说明 |
| --- | --- |
| 1 | **static List<Throwable> getCausalChain(Throwable throwable)** 获取一个Throwable的原因链的列表。 |
| 2 | **static Throwable getRootCause(Throwable throwable)** 返回抛出的最里面的原因。 |
| 3 | **static String getStackTraceAsString(Throwable throwable)** 返回包含toString()的结果字符串，随后完整抛出，递归的堆栈跟踪。 |
| 4 | **static RuntimeException propagate(Throwable throwable)** 传播抛出原样如果RuntimeException或Error是一个实例，否则作为最后的报告，把它包装在一个RuntimeException，然后传播。 |
| 5 | **static <X extends Throwable> void propagateIfInstanceOf(Throwable throwable, Class<X> declaredType)** 传播抛出对象完全按原样，当且仅当它是declaredType的一个实例。 |
| 6 | **static void propagateIfPossible(Throwable throwable)** 传播抛出对象完全按原样，当且仅当它是RuntimeException或Error的一个实例。 |
| 7 | **static <X extends Throwable> void propagateIfPossible(Throwable throwable, Class<X> declaredType)** 传播抛出对象完全按原样，当且仅当它是RuntimeException，错误或的declaredType的一个实例。 |
| 8 | **static <X1 extends Throwable,X2 extends Throwable>void propagateIfPossible(Throwable throwable, Class<X1> declaredType1, Class<X2> declaredType2)** 传播抛出对象完全按原样，当且仅当它是RuntimeException，Error，declaredType1或declaredType2的一个实例。 |

## 继承的方法

这个类继承了以下类方法：

- java.lang.Object

# Throwables示例

创建使用所选择的任何编辑器下面的java程序，比如 **C:/> Guava**

*GuavaTester.java*

```
import java.io.IOException;

import com.google.common.base.Objects;
```

```
import com.google.common.base.Throwables;

public class GuavaTester {
   public static void main(String args[]){
      GuavaTester tester = new GuavaTester();
      try {
         tester.showcaseThrowables();
      } catch (InvalidInputException e) {
         //get the root cause
         System.out.println(Throwables.getRootCause(e));
      }catch (Exception e) {
         //get the stack trace in string format
         System.out.println(Throwables.getStackTraceAsString(e));
      }

      try {
         tester.showcaseThrowables1();
      }catch (Exception e) {
         System.out.println(Throwables.getStackTraceAsString(e));
      }
   }

   public void showcaseThrowables() throws InvalidInputException{
      try {
         sqrt(-3.0);
      } catch (Throwable e) {
         //check the type of exception and throw it
         Throwables.propagateIfInstanceOf(e, InvalidInputException.
         Throwables.propagate(e);
      }
   }

   public void showcaseThrowables1(){
      try {
         int[] data = {1,2,3};
         getValue(data, 4);
      } catch (Throwable e) {
         Throwables.propagateIfInstanceOf(e, IndexOutOfBoundsExcept
         Throwables.propagate(e);
      }
   }

   public double sqrt(double input) throws InvalidInputException{
      if(input < 0) throw new InvalidInputException();
      return Math.sqrt(input);
   }

   public double getValue(int[] list, int index) throws IndexOutOfE
      return list[index];
   }

   public void dummyIO() throws IOException {
      throw new IOException();
```

```
    }
}

class InvalidInputException extends Exception {
}
```

# 验证结果

使用javac编译器编译如下类

```
C:\Guava>javac GuavaTester.java
```

现在运行GuavaTester看到的结果

```
C:\Guava>java GuavaTester
```

看到结果。

```
InvalidInputException
java.lang.ArrayIndexOutOfBoundsException: 4
    at GuavaTester.getValue(GuavaTester.java:52)
    at GuavaTester.showcaseThrowables1(GuavaTester.java:38)
    at GuavaTester.main(GuavaTester.java:19)
```

# Guava集合工具 - Guava教程

Guava介绍了基于开发者的应用开发经验，许多先进的集合。以下是有用的集合列表：

| S.N. | 集合名称和说明 |
|---|---|
| 1 | Multiset 一个扩展来设置界面，允许重复的元素。 |
| 2 | Multimap 一个扩展来映射接口，以便其键可一次被映射到多个值 |
| 3 | BiMap 一个扩展来映射接口，支持反向操作。 |
| 4 | Table 表代表一个特殊的图，其中两个键可以在组合的方式被指定为单个值。 |

# Guava缓存工具 - Guava教程

Guava通过接口LoadingCache提供了一个非常强大的基于内存的LoadingCache<K，V>。在缓存中自动加载值，它提供了许多实用的方法，在有缓存需求时非常有用。

## 接口声明

以下是forcom.google.common.cache.LoadingCache<K，V>接口的声明：

```
@Beta
@GwtCompatible
public interface LoadingCache<K,V>
   extends Cache<K,V>, Function<K,V>
```

## 接口方法

| S.N. | 方法及说明 |
| --- | --- |
| 1 | **V apply(K key)** 不推荐使用。提供满足功能接口;使用get(K)或getUnchecked(K)代替。 |
| 2 | **ConcurrentMap<K,V> asMap()** 返回存储在该缓存作为一个线程安全的映射条目的视图。 |
| 3 | **V get(K key)** 返回一个键在这个高速缓存中，首先装载如果需要该值相关联的值。 |
| 4 | **ImmutableMap<K,V> getAll(Iterable<? extends K> keys)** 返回一个键相关联的值的映射，创建或必要时检索这些值。 |
| 5 | **V getUnchecked(K key)** 返回一个键在这个高速缓存中，首先装载如果需要该值相关联的值。 |
| 6 | **void refresh(K key)** 加载键key，可能是异步的一个新值。 |

# LoadingCache 示例

使用所选择的编辑器创建下面的java程序 **C:/> Guava**

*GuavaTester.java*

```
import java.util.HashMap;
```

```
import java.util.Map;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;

import com.google.common.base.MoreObjects;
import com.google.common.cache.CacheBuilder;
import com.google.common.cache.CacheLoader;
import com.google.common.cache.LoadingCache;

public class GuavaTester {
   public static void main(String args[]){
      //create a cache for employees based on their employee id
      LoadingCache <string, employee="" style="box-sizing: border-b
         CacheBuilder.newBuilder()
            .maximumSize(100) // maximum 100 records can be cached
            .expireAfterAccess(30, TimeUnit.MINUTES) // cache will
            .build(new CacheLoader<string, employee="" style="box-s
               @Override
               public Employee load(String empId) throws Exception
                  //make the expensive call
                  return getFromDatabase(empId);
               }
            });

      try {
         //on first invocation, cache will be populated with corres
         //employee record
         System.out.println("Invocation #1");
         System.out.println(employeeCache.get("100"));
         System.out.println(employeeCache.get("103"));
         System.out.println(employeeCache.get("110"));
         //second invocation, data will be returned from cache
         System.out.println("Invocation #2");
         System.out.println(employeeCache.get("100"));
         System.out.println(employeeCache.get("103"));
         System.out.println(employeeCache.get("110"));

      } catch (ExecutionException e) {
         e.printStackTrace();
      }
   }

   private static Employee getFromDatabase(String empId){
      Employee e1 = new Employee("Mahesh", "Finance", "100");
      Employee e2 = new Employee("Rohan", "IT", "103");
      Employee e3 = new Employee("Sohan", "Admin", "110");

      Map <string, employee="" style="box-sizing: border-box;">data
      database.put("100", e1);
      database.put("103", e2);
      database.put("110", e3);
      System.out.println("Database hit for" + empId);
      return database.get(empId);
```

```
   }
}

class Employee {
   String name;
   String dept;
   String emplD;

   public Employee(String name, String dept, String empID){
      this.name = name;
      this.dept = dept;
      this.emplD = empID;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getDept() {
      return dept;
   }
   public void setDept(String dept) {
      this.dept = dept;
   }
   public String getEmplD() {
      return emplD;
   }
   public void setEmplD(String emplD) {
      this.emplD = emplD;
   }

   @Override
   public String toString() {
      return MoreObjects.toStringHelper(Employee.class)
      .add("Name", name)
      .add("Department", dept)
      .add("Emp Id", emplD).toString();
   }
}</string,></string,></string,></string,>
```

## 验证结果

使用javac编译器如下编译类

```
C:\Guava>javac GuavaTester.java
```

现在运行GuavaTester看到的结果

```
C:\Guava>java GuavaTester
```

看看结果：

```
Invocation #1
Database hit for100
Employee{Name=Mahesh, Department=Finance, Emp Id=100}
Database hit for103
Employee{Name=Rohan, Department=IT, Emp Id=103}
Database hit for110
Employee{Name=Sohan, Department=Admin, Emp Id=110}
Invocation #2
Employee{Name=Mahesh, Department=Finance, Emp Id=100}
Employee{Name=Rohan, Department=IT, Emp Id=103}
Employee{Name=Sohan, Department=Admin, Emp Id=110}
```

# Guava字符串工具 - Guava教程

Guava介绍基于开发者的应用开发经验，许多先进的字符串工具。以下是有用的基于字符串的实用程序列表：

| S.N. | 程序名称和说明 |
|---|---|
| 1 | Joiner 实用加入对象，字符串等。 |
| 2 | Spilter 实用程序用来分割字符串。 |
| 3 | CharMatcher 实用的字符操作。 |
| 4 | CaseFormat 实用程序，用于改变字符串格式。 |

# Guava原语工具 - Guava教程

作为Java的原语类型，不能用来传递在泛型或于类别作为输入。Guava提供大量包装工具类来处理原始类型的对象。以下是有用的原始处理工具的列表：

| S.N. | 程序名称和说明 |
|---|---|
| 1 | Bytes 实用程序的原始字节。 |
| 2 | Shorts 实用的原始short。 |
| 3 | Ints 实用为基本整型。 |
| 4 | Longs 实用的原始长整型。 |
| 5 | Floats 实用为基本float。 |
| 6 | Doubles 实用为基本的double。 |
| 7 | Chars 实用的原始字符。 |
| 8 | Booleans 实用为基本布尔。 |

# Guava数学工具 - Guava教程

提供Guava数学相关的实用工具类来处理int，long及BigInteger。以下是有用的工具列表：

| S.N. | 程序名称和说明 |
| --- | --- |
| 1 | IntMath 数学工具为int类型。 |
| 2 | LongMath 数学工具为long类型。 |
| 3 | BigIntegerMath 数学实用程序处理BigInteger。 |

# Hibernate 教程

Hibernate是一个高性能的对象/关系持久性和其基于开源GNU宽通用公共许可证（LGPL）授权，可以免费下载查询服务。 Hibernate不仅仅关心从Java类映射到数据库表（包括Java数据类型到SQL数据类型），还提供数据查询和获取。

本教程将教你如何使用Hibernate来开发简单基于数据库的Web应用程序。

# 读者

本教程是为Java程序员设计了一个需要了解Hibernate框架和API。完成本教程后，你会发现自己在专业知识的中等水平。

# 必备条件

我们假设你有Java编程语言的很好的理解。关系数据库的一个基本的了解，JDBC和SQL是非常有帮助的。

# ORM是什么？ORM介绍 - hibernate

## JDBC是什么?

JDBC代表Java数据库连接，并提供一组Java API，用于Java程序访问关系数据库。这些Java的API允许Java程序执行SQL语句，并与任何SQL兼容的数据库进行交互。

JDBC提供了一个灵活的架构来编写一个独立于数据库应用程序，它可以在不同的平台上运行，并与不同的数据库管理系统交互，而无需任何修改。

## JDBC的优点和缺点

| JDBC的优点 | JDBC的缺点 |
| --- | --- |
| 干净和简单的SQL处理 良好的性能与大数据 用于小型应用很不错 简单，语法很容易学习 | 复杂，如果它被用在大型工程项目 大的编程开销 无封装 难以实现MVC的概念 查询是具体的数据库管理系统 |

## 为什么用对象关系映射（ORM）？

当我们与一个面向对象的系统工作，还有的对象模型和关系数据库之间的不匹配。RDBMS代表以表格格式数据，而面向对象的语言，如Java或C＃将其表示为对象的互连图。考虑下面的Java类与适当的构造函数和相关的公共功能：

```
public class Employee {
   private int id;
   private String first_name;
   private String last_name;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.first_name = fname;
      this.last_name = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public String getFirstName() {
      return first_name;
   }
   public String getLastName() {
      return last_name;
   }
   public int getSalary() {
      return salary;
   }
}
```

考虑上述目的需要存储和检索到下面的RDBMS表：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

第一个问题，如果我们需要开发有几页或我们的应用程序后，修改数据库的设计？其次，加载并存储对象在关系数据库使我们面临以下五大不匹配的问题。

| 不匹配 | 描述 |
|---|---|
| Granularity | 有时候，将有哪些具有比在数据库中对应的表的数目更多的类的对象模型。 |
| Inheritance | RDBMS中没有定义任何类似继承是面向对象编程语言的自然范式。 |
| Identity | 关系数据库定义的'千篇一律'只有一个概念：主键。但是，Java定义了对象标识（a== b）和对象相等（那么a.Equals（b）） |
| Associations | 面向对象的语言表示使用的对象引用，其中的上午RDBMS代表一个协会作为一个外键列关联。 |
| Navigation | 访问Java和关系数据库对象的方法是根本不同。 |

对象 - 关系映射（ORM）是解决处理所有上述阻抗失配。

# 什么是ORM？

ORM代表对象 - 关系映射（ORM）是一种编程技术的ORM系统已通过纯JDBC的优点如下关系数据库和面向对象的编程语言，如Java，C＃等之间转换数据

| S.N. | 优点 |
|---|---|
| 1 | 让业务代码访问对象，而不是数据库表。 |
| 2 | 隐藏了面向对象的逻辑SQL查询详情。 |
| 3 | 基于JDBC的“引擎盖下” |
| 4 | 无需处理数据库实现。 |
| 5 | 基于业务概念，而不是数据库结构的实体。 |
| 6 | 事务管理和自动密钥生成。 |
| 7 | 应用程序的快速开发。 |

ORM解决方案由以下四种实体：

| S.N. | 解决 |
|---|---|
| 1 | 一个API来对持久化类的对象执行基本的CRUD操作。 |
| 2 | 语言或API来指定引用的类和类的属性查询。 |
| 3 | 一个可配置的设备，用于指定映射元数据。 |
| 4 | 技术与事务对象交互，以执行脏数据检查，懒关联加载，以及其他优化功能。 |

## Java ORM框架：

有几种持久性框架和Java的ORM方案。持久性框架是一个ORM的服务，存储和检索对象到关系型数据库。

- 企业JavaBeans实体Bean
- Java Data Objects
- Castor
- TopLink
- Spring DAO
- Hibernate
- And many more

# Hibernate概述,Hibernate是什么？ - hibernate

Hibernate是对Java中的对象关系映射（ORM）解决方案，它由加文·金（Gavin King）在2001年提出并创建的一个开源持久框架。它是一个强大的，高性能的对象关系持久性和对任何Java应用程序的查询服务。

Hibernate映射Java类到数据库表和从Java数据类型到SQL数据类型和95%的通用数据持久化相关的编程任务，解放了开发者。

Hibernate位于传统的Java对象和数据库服务器来处理在持久化的基础上，适当的O/R机制和模式，这些对象在所有工作对象之间。




## Hibernate 优点:

- Hibernate会处理映射的Java类来使用XML文件，数据库表和无需编写任何一行代码。
- 提供了简单的API，用于直接从数据库中存储和检索Java对象。
- 如果有变化，数据库或任何表中的那么只需要修改XML文件的属性。
- 抽象掉不熟悉的SQL类型，并提供我们解决熟悉的Java对象。
- Hibernate不要求应用服务器进行操作。
- 操纵数据库对象的复杂关联。
- 尽量减少与智能读取策略数据库的访问。
- 提供数据的简单查询。

## 支持的数据库 ：

Hibernate支持几乎所有主要的RDBMS。以下是Hibernate支持的几个数据库引擎列表。

- HSQL Database Engine
- DB2/NT

- MySQL
- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database
- Sybase SQL Server
- Informix Dynamic Server

## 支持的技术：

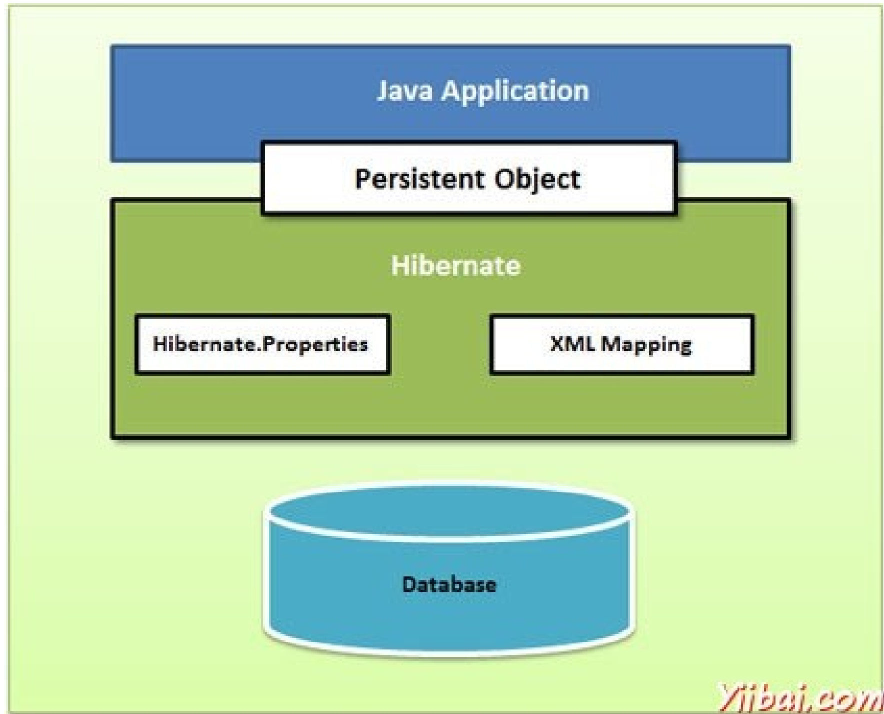
Hibernate支持各种各样的其他技术，包括以下内容：

- XDoclet Spring
- J2EE
- Eclipse plug-ins
- Maven

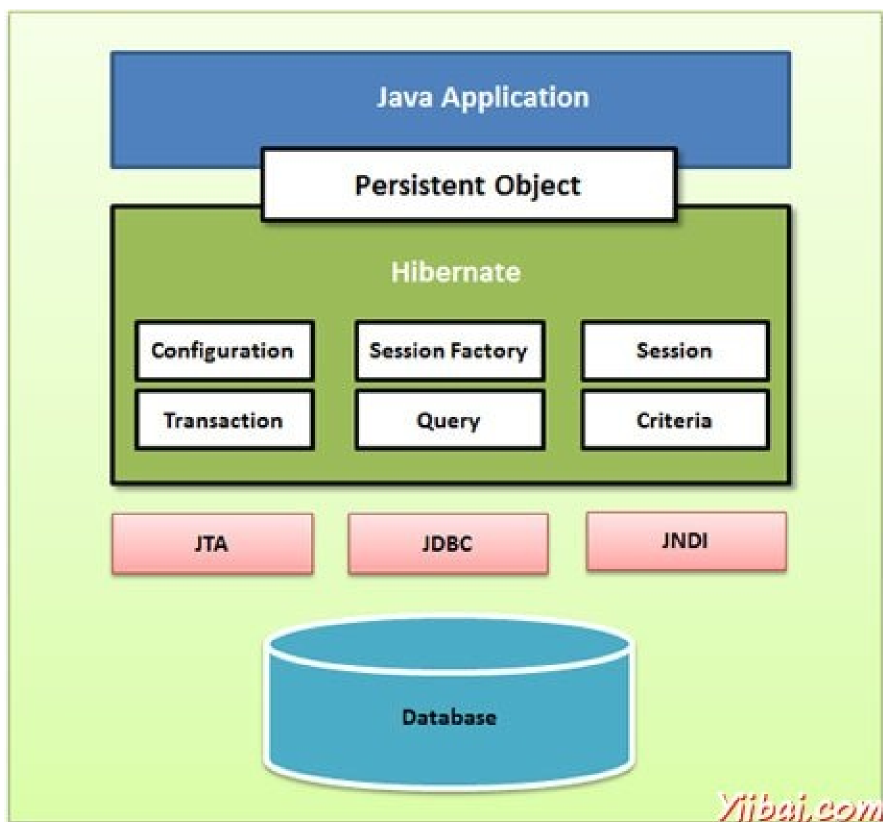
# Hibernate架构 - hibernate

Hibernate架构是分层的，隔离的不必知道底层API。 Hibernate中使用数据库和配置信息来为应用程序提供持久化服务（以及持久的对象）。

下面是Hibernate应用程序体系结构的一个非常高的水平视图。




下面是Hibernate的应用架构与一些重要的核心课程的详细视图。




Hibernate使用各种现有的Java API，如JDBC，Java事务API（JTA）和Java命名和目录接口（JNDI）。 JDBC提供了常见的关系数据库功能的抽象的一个基本水平，使具有JDBC驱动程序，Hibernate的支持几乎任何数据库。 JNDI和JTA允许Hibernate与J2EE应用服务器进行集成。

以下部分列出了每个参与Hibernate应用程序体系结构的类的对象的简要说明。

## Configuration 对象:

Configuration对象是你在任何Hibernate应用程序中创建并通常在应用程序初始化创建一次，第一个Hibernate的对象。它代表了Hibernate所需的配置或属性文件。Configuration对象提供了两个按键组成部分：

- 数据库连接：这是通过Hibernate支持的一个或多个配置文件来处理。这些文件是：hibernate.properties和hibernate.cfg.xml。
- 类映射设置 此组件创建Java类和数据库表之间的连接

## SessionFactory 对象:

Configuration对象用于创建一个SessionFactory对象，它反过来可以配置Hibernate的使用提供的配置文件的应用程序，并允许一个Session对象被实例化。通过SessionFactory是线程安全的对象和使用的应用程序的所有线程。

通过SessionFactory是重量级的对象，因此通常它被应用程序时创建的启动和保持以备后用。将使用一个单独的配置文件需要每个数据库都有一个SessionFactory对象。所以，如果正在使用多个数据库，那么就需要创建多个SessionFactory的对象。

## Session 对象:

Session对象用于获取与数据库的物理连接。 Session对象是重量轻，设计了一个互动是需要与数据库每次被实例化。持久化对象被保存，并通过一个Session对象中检索。

会话中的对象不应该保持开放很长一段时间，因为他们通常不被线程安全的，应该被创建并根据需要销毁他们。

## Transaction 对象:

事务代表一个工作单元与数据库和大部分RDBMS支持事务功能。在Hibernate事务是由一个基本的事务管理器和事务（从JDBC或JTA）来处理。

这是一个可选的对象和Hibernate应用程序可以选择不使用这个接口，而不是在他们自己的应用程序代码管理事务。

## Query 对象:

查询对象使用SQL或Hibernate查询语言（HQL）字符串从数据库中检索数据并创建对象。一个查询实例是用来绑定查询参数，限制查询返回的结果数量，并最终执行查询。

## Criteria 对象:

Criteria对象用于创建和执行面向对象的条件查询来检索对象。

# Hibernate环境配置 - hibernate

本章将解释如何安装Hibernate和其他相关的包准备开发环境为Hibernate应用程序。我们将使用MySQL数据库的工作，尝试使用Hibernate的例子，所以一定要确保已经安装的MySQL数据库。有关MySQL的一个更详细信息，可以查看MySQL教程.

## 下载Hibernate:

假定你已经拥有Java的最新版本安装在机器上。以下是简单的步骤下载并安装Hibernate。

- 选择是否要在Windows或UNIX安装Hibernate，然后继续下一个步骤，下载zip文件适用于Windows，而Unix则为.tz文件。
- 下载Hibernate最新版本 http://www.hibernate.org/downloads.
- 在写这篇教程的时候，我下载了hibernate-distribution-3.6.4.Final，解压缩下载的文件，目录结构如下所示。

| Name | Date modified | Type | Size |
|---|---|---|---|
| documentation | 04/05/2011 15:16 | File folder | |
| lib | 04/05/2011 15:15 | File folder | |
| project | 04/05/2011 14:44 | File folder | |
| changelog.txt | 04/05/2011 14:44 | Text Document | 224 KB |
| hibernate_logo.gif | 04/05/2011 14:44 | GIF image | 2 KB |
| hibernate3.jar | 04/05/2011 15:15 | Executable Jar File | 4,037 KB |
| hibernate-testing.jar | 04/05/2011 15:03 | Executable Jar File | 44 KB |
| lgpl.txt | 04/05/2011 14:44 | Text Document | 26 KB |



## 安装Hibernate:

下载并解压Hibernate安装最新版本，需要执行下面两个简单的步骤。请确保你设置CLASSPATH变量正确，否则编译应用程序将面临问题。

- 现在所有的库文件拷贝从 /lib 目录到CLASSPATH，并改变classpath变量包括所有的JAR文件：
- 最后复制 hibernate3.jar 里的文件到CLASSPATH。该文件位于安装的根目录，Hibernate需要做的工作主要的JAR。

## Hibernate Prerequisites:

以下是由Hibernate要求，应该开始使用Hibernate之前安装它们的软件包 /lib 列表。要安装这些软件包，从/lib目录拷贝库文件到CLASSPATH，并相应地改变CLASSPATH变量。

| S.N. | Packages/Libraries |
|---|---|
| 1 | **dom4j** - XML parsing www.dom4j.org/ |
| 2 | **Xalan** - XSLT Processor http://xml.apache.org/xalan-j/ |
| 3 | **Xerces** - The Xerces Java Parser http://xml.apache.org/xerces-j/ |
| 4 | **cglib** - Appropriate changes to Java classes at runtime http://cglib.sourceforge.net/ |
| 5 | **log4j** - Logging Faremwork http://logging.apache.org/log4j |
| 6 | **Commons** - Logging, Email etc. http://jakarta.apache.org/commons |
| 7 | **SLF4J** - Logging Facade for Java http://www.slf4j.org |

# Hibernate配置 - hibernate

Hibernate要求预先知道在哪里可以找到它定义Java类是如何关联到数据库表的映射信息。 Hibernate也需要一组相关的数据库和其他相关参数的配置设置。所有这些信息通常是hibernate.properties，一个标准的Java属性文件，或者作为一个名为hibernate.cfg.xml的XML文件。

考虑XML格式的文件hibernate.cfg.xml中指定在例子必需的Hibernate属性。大部分的属性取默认值，它不需要在属性文件中指定它们，除非它真的是必需的。此文件保存在应用程序的类路径的根目录。

## Hibernate 属性:

以下是需要在独立情况下配置一个数据库的重要属性的列表：

| S.N. | 属性和说明 |
| --- | --- |
| 1 | **hibernate.dialect** This property makes Hibernate generate the appropriate SQL for the chosen database. |
| 2 | **hibernate.connection.driver_class** The JDBC driver class. |
| 3 | hibernate.connection.url The JDBC URL to the database instance. |
| 4 | hibernate.connection.username The database username. |
| 5 | hibernate.connection.password The database password. |
| 6 | hibernate.connection.pool_size Limits the number of connections waiting in the Hibernate database connection pool. |
| 7 | hibernate.connection.autocommit Allows autocommit mode to be used for the JDBC connection. |

如果是随着应用程序服务器和JNDI使用一个数据库，那么就必须配置以下属性：

| S.N. | 属性和说明 |
| --- | --- |
| 1 | **hibernate.connection.datasource** The JNDI name defined in the application server context you are using for the application. |
| 2 | **hibernate.jndi.class** The InitialContext class for JNDI. |
| 3 | **hibernate.jndi.<JNDIpropertyname>** Passes any JNDI property you like to the JNDI *InitialContext*. |
| 4 | **hibernate.jndi.url** Provides the URL for JNDI. |
| 5 | **hibernate.connection.username** The database username. |
| 6 | **hibernate.connection.password** The database password. |

## Hibernate和MySQL数据库：

MySQL是最受欢迎的开源数据库系统之一。让我们创建hibernate.cfg.xml配置文件，并将其放置在应用程序的类的根路径。必须确保有MySQL数据库可供TESTDB数据库，必须提供一个用户test来访问数据库。

XML配置文件必须符合Hibernate 3配置的DTD，可从 http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
   <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
   </property>
   <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
   </property>

   <!-- Assume test is the database name -->
   <property name="hibernate.connection.url">
      jdbc:mysql://localhost/test
   </property>
   <property name="hibernate.connection.username">
      root
   </property>
   <property name="hibernate.connection.password">
      root123
   </property>

   <!-- List of XML mapping files -->
   <mapping resource="Employee.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

上面的配置文件包含这都与hibernate映射文件<mapping>标签，我们将在下一章看到到底什么是Hibernate映射文件，以及如何和为什么要使用它。以下是各种重要的数据库方言属性类型的列表：

| Database | Dialect Property |
| --- | --- |
| DB2 | org.hibernate.dialect.DB2Dialect |
| HSQLDB | org.hibernate.dialect.HSQLDialect |
| HypersonicSQL | org.hibernate.dialect.HSQLDialect |
| Informix | org.hibernate.dialect.InformixDialect |
| Ingres | org.hibernate.dialect.IngresDialect |
| Interbase | org.hibernate.dialect.InterbaseDialect |
| Microsoft SQL Server 2000 | org.hibernate.dialect.SQLServerDialect |
| Microsoft SQL Server 2005 | org.hibernate.dialect.SQLServer2005Dialect |
| Microsoft SQL Server 2008 | org.hibernate.dialect.SQLServer2008Dialect |
| MySQL | org.hibernate.dialect.MySQLDialect |
| Oracle (any version) | org.hibernate.dialect.OracleDialect |
| Oracle 11g | org.hibernate.dialect.Oracle10gDialect |
| Oracle 10g | org.hibernate.dialect.Oracle10gDialect |
| Oracle 9i | org.hibernate.dialect.Oracle9iDialect |
| PostgreSQL | org.hibernate.dialect.PostgreSQLDialect |
| Progress | org.hibernate.dialect.ProgressDialect |
| SAP DB | org.hibernate.dialect.SAPDBDialect |
| Sybase | org.hibernate.dialect.SybaseDialect |
| Sybase Anywhere | org.hibernate.dialect.SybaseAnywhereDialect |

# Hibernate Sessions - hibernate

Session对象用于获取与数据库的物理连接。 Session对象是重量轻，设计了一个互动是需要与数据库每次被实例化。持久化对象被保存，并通过一个Session对象中检索。

会话中的对象不应该保持开放很长一段时间，因为他们通常不被线程安全的，他们应该被创建并根据需要摧毁他们。这次会议的主要功能是提供创建，读取和删除操作映射的实体类的实例。实例中可能存在以下三种状态之一在给定时间点：

- 短暂性: 持久化类的未与会话相关联，并在数据库中没有代表性，没有标识值的新实例被Hibernate认为是暂时的。
- 持久性: 可以做一个瞬态的实例持久化通过将它与一个会话相关联。持久性实例都有一个表示在数据库中，一个标识符值，与会话相关联。
- 独立性: 一旦我们关闭Hibernate的Session，持久化实例将成为一个分离的实例。

一个Session实例是可序列化的，如果它的持久化类是可序列化的。一个典型的事务应该使用下面的语句：

```
Session session = factory.openSession();
Transaction tx = null;
try {
   tx = session.beginTransaction();
   // do some work
   ...
   tx.commit();
}
catch (Exception e) {
   if (tx!=null) tx.rollback();
   e.printStackTrace();
}finally {
   session.close();
}
```

如果Session抛出异常，事务必须回滚，会话必须被丢弃。

## Session 接口方法:

Session接口提供多个方法，但这里列出的只有少数重要的方法，这些方法我们在本教程中会使用。您可以查看Hibernate文档Session和SessionFactory相关方法的完整列表。

| S.N. | 会话的方法和说明 |
|---|---|

| 1 | **Transaction beginTransaction()** Begin a unit of work and return the associated Transaction object. |
|---|---|
| 2 | **void cancelQuery()** Cancel the execution of the current query. |
| 3 | **void clear()** Completely clear the session. |
| 4 | **Connection close()** End the session by releasing the JDBC connection and cleaning up. |
| 5 | **Criteria createCriteria(Class persistentClass)** Create a new Criteria instance, for the given entity class, or a superclass of an entity class. |
| 6 | **Criteria createCriteria(String entityName)** Create a new Criteria instance, for the given entity name. |
| 7 | **Serializable getIdentifier(Object object)** Return the identifier value of the given entity as associated with this session. |
| 8 | **Query createFilter(Object collection, String queryString)** Create a new instance of Query for the given collection and filter string. |
| 9 | **Query createQuery(String queryString)** Create a new instance of Query for the given HQL query string. |
| 10 | **SQLQuery createSQLQuery(String queryString)** Create a new instance of SQLQuery for the given SQL query string. |
| 11 | **void delete(Object object)** Remove a persistent instance from the datastore. |
| 12 | **void delete(String entityName, Object object)** Remove a persistent instance from the datastore. |
| 13 | **Session get(String entityName, Serializable id)** Return the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance. |
| 14 | **SessionFactory getSessionFactory()** Get the session factory which created this session. |
| 15 | **void refresh(Object object)** Re-read the state of the given instance from the underlying database. |
| 16 | **Transaction getTransaction()** Get the Transaction instance associated with this session. |
| 17 | **boolean isConnected()** Check if the session is currently connected. |
| 18 | **boolean isDirty()** Does this session contain any changes which must be synchronized with the database? |
| 19 | **boolean isOpen()** Check if the session is still open. |
|  | **Serializable save(Object object)** Persist the given transient instance, |

| | |
|---|---|
| | first assigning a generated identifier. |
| 21 | **void saveOrUpdate(Object object)** Either save(Object) or update(Object) the given instance. |
| 22 | **void update(Object object)** Update the persistent instance with the identifier of the given detached instance. |
| 23 | **void update(String entityName, Object object)** Update the persistent instance with the identifier of the given detached instance. |

# Hibernate持久化类 - hibernate

Hibernate的整个概念是采取从Java类属性的值，并将持久到数据库表。一个映射文件Hibernate的帮助确定如何从拉动类的值，并将它们映射与表和相关的域。

其对象或实例将存储在数据库表中的Java类在Hibernate中称为持久化类。Hibernate的效果最好，如果这些类遵循一些简单的规则，也称为普通Java对象（POJO）编程模型。有下列持久化类的主要规则，但是，这些规则并不是必需的。

- 将所有的持久化Java类需要一个默认的构造函数。
- 所有类应该包含为了让容易识别对象内Hibernate和数据库的ID。此属性映射到数据库表的主键列。
- 所有属性将被持久化应该声明为private，并已在JavaBean风格的定义的getXXX和setXXX方法。
- Hibernate的关键功能，代理，取决于持久化类或者是非final的，或者说声明的所有公共方法的接口的实现。
- 所有的类不扩展或实现的EJB框架需要进行一些专门的类和接口。

POJO名称用于强调一个给定的对象是一个普通的Java对象，而不是一个特殊的对象，好更不是Enterprise JavaBean。

## 一个简单的POJO例子：

基于上面提到的一些规则，我们可以如下定义一个POJO类：

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

# Hibernate映射文件 - hibernate

对象/关系映射的XML文档中通常被定义。这个映射文件指示Hibernate如何定义的一个或多个类映射到数据库表。

虽然很多Hibernate用户选择手工编写XML中，有一些工具可以用来生成映射文档。包括XDoclet，Middlegen和AndroMDA等用于高级Hibernate的用户。

让我们考虑我们的对象将坚持在下一节中定义的表前面定义的POJO类。

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

会有一个表对应于每一个对象，你愿意提供持久性。考虑上述目的需要存储和检索到下面的RDBMS表：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

基于以上两个实体，我们可以定义它指示Hibernate如何定义的一个或多个类映射到数据库表下面的映射文件。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string",
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

保存的映射文件中的格式： <classname>.hbm.xml。我们保存映射文件中的文件Employee.hbm.xml。来看看关于在映射文件中使用的映射元素的小细节：

- 映射文档是具有<hibernate-mapping>为包含所有的<class>元素的根元素的XML文档。
- 在<class>元素被用于定义数据库表从一个Java类特定的映射。 Java类名指定使用class元素的name属性和使用表属性数据库表名指定。
- <meta>元素是可选元素，可以用来创建类的描述。
- <id>元素映射在类中的唯一ID属性到数据库表的主键。 id元素的name属性是指属性的类和column属性是指在数据库表中的列。 type属性保存了Hibernate映射类型，这种类型的映射将会从Java转换为SQL数据类型。

- id元素内的<generator>元素被用来自动生成的主键值。将生成元素的class属性设置为原生让Hibernate拿起无论是identity，sequence或者hilo中的算法来创建主键根据底层数据库的支持能力。
- <property>元素用于一个Java类的属性映射到数据库表中的列。元素的name属性是指属性的类和column属性是指在数据库表中的列。 type属性保存了Hibernate映射类型，这种类型的映射将会从Java转换为SQL数据类型。

还有这将在映射文件中使用，接下来尽量覆盖尽可能多其他的Hibernate相关主题的其他属性和可用的元素。

# Hibernate映射类型 - hibernate

当编写Hibernate映射文件，映射的Java数据类型到关系型数据库的数据类型。声明及在映射文件中使用的类型不是Java的数据类型，它们不是SQL数据库的数据类型。这些类型就是所谓的Hibernate映射类型，它可以从Java转换到SQL数据类型，反之亦然。

本章列出了所有的基本，日期和时间，大对象，以及其他各种内置映射类型。

## 基本类型：

| 映射类型 | Java 类型 | ANSI SQL 类型 |
|---|---|---|
| integer | int or java.lang.Integer | INTEGER |
| long | long or java.lang.Long | BIGINT |
| short | short or java.lang.Short | SMALLINT |
| float | float or java.lang.Float | FLOAT |
| double | double or java.lang.Double | DOUBLE |
| big_decimal | java.math.BigDecimal | NUMERIC |
| character | java.lang.String | CHAR(1) |
| string | java.lang.String | VARCHAR |
| byte | byte or java.lang.Byte | TINYINT |
| boolean | boolean or java.lang.Boolean | BIT |
| yes/no | boolean or java.lang.Boolean | CHAR(1) ('Y' or 'N') |
| true/false | boolean or java.lang.Boolean | CHAR(1) ('T' or 'F') |

## Date 和time 类型:

| Mapping 类型 | Java 类型 | ANSI SQL 类型 |
|---|---|---|
| date | java.util.Date or java.sql.Date | DATE |
| time | java.util.Date or java.sql.Time | TIME |
| timestamp | java.util.Date or java.sql.Timestamp | TIMESTAMP |
| calendar | java.util.Calendar | TIMESTAMP |
| calendar_date | java.util.Calendar | DATE |

## 二进制大对象类型：

| Mapping 类型 | Java 类型 | ANSI SQL 类型 |
|---|---|---|
| binary | byte[] | VARBINARY (or BLOB) |
| text | java.lang.String | CLOB |
| serializable | any Java class that implements java.io.Serializable | VARBINARY (or BLOB) |
| clob | java.sql.Clob | CLOB |
| blob | java.sql.Blob | BLOB |

## JDK-related 类型:

| Mapping 类型 | Java 类型 | ANSI SQL 类型 |
|---|---|---|
| class | java.lang.Class | VARCHAR |
| locale | java.util.Locale | VARCHAR |
| timezone | java.util.TimeZone | VARCHAR |
| currency | java.util.Currency | VARCHAR |

# Hibernate实例 - hibernate

我们尝试使用Hibernate提供的一个独立应用程序Java持久化例子。通过使用Hibernate技术创建Java应用程序，步骤如下：

## 创建POJO 类:

在创建应用程序的第一步是建立在Java POJO类或类，具体取决于将被持久化到数据库的应用程序。让我们考虑我们的Employee类的getXXX和setXXX方法，使其兼容JavaBean类。

POJO（普通Java对象）是一个Java对象，它不扩展或实现分别需要由EJB框架一些专门的类和接口。所有普通的Java对象都是POJO。

当设计一个类里面要Hibernate持久化，重要的是要提供的JavaBeans兼容的代码以及将工作作为索引，就像在Employee类的id属性。

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

## 创建数据库表:

第二步是在数据库中创建表。会有一张表对应于每一个对象提供持久性。考虑上述目的需要存储和检索到下面的RDBMS表：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

## 创建映射配置文件：

这一步是创建一个指示Hibernate如何定义的一个或多个类映射到数据库表的映射文件。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

应该保存的映射文件中的格式**<classname>.hbm.xml**。我们保存我们的映射文件中的文件Employee.hbm.xml。让我们来看看有关映射文件的小细节：

- 映射文档是具有**<hibernate-mapping>**为包含所有的<class>元素的根元素的XML文档。
- **<class>**元素被用于定义数据库表从一个Java类特定的映射。 Java类名指定使用class元素的name属性和使用表属性数据库表名指定。
- **<meta>**元素是可选元素，可以用来创建类的描述。
- **<id>**元素映射在类中的唯一ID属性到数据库表的主键。 id 元素的 name 属性是指属性的类和 column 属性是指在数据库表中的列。 type属性保存了Hibernate映射类型，这种类型的映射将会从Java转换为SQL数据类型。

- **id** 元素内的 **<generator>** 元素被用来自动生成的主键值。将生成元素的class属性设置为原产于让Hibernate无论是identity，sequence或者hilo中的算法来创建主键根据底层数据库的支持能力。
- **<property>** 元素用于一个Java类的属性映射到数据库表中的列。元素的name属性是指属性的类和column属性是指在数据库表中的列。 type属性保存了Hibernate映射类型，这种类型的映射将会从Java转换为SQL数据类型。

还有这将在映射文件中使用，我会尽量覆盖尽可能多的在讨论其他的Hibernate相关主题的其他属性和可用的元素。

## 创建应用程序类：

最后，我们将创建应用程序类 main() 方法来运行应用程序。我们将使用这个应用程序来保存一些雇员的记录，然后我们将申请CRUD操作上的记录。

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try{
         factory = new Configuration().configure().buildSessionFact
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object
         throw new ExceptionInInitializerError(ex);
      }
      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 10000);

      /* List down all the employees */
      ME.listEmployees();

      /* Update employee's records */
      ME.updateEmployee(empID1, 5000);

      /* Delete an employee from the database */
      ME.deleteEmployee(empID2);
```

```
      /* List down new list of the employees */
      ME.listEmployees();
   }
   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salar
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return employeeID;
   }
   /* Method to  READ all the employees */
   public void listEmployees( ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         List employees = session.createQuery("FROM Employee").list
         for (Iterator iterator =
                           employees.iterator(); iterator.hasNext(
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName
            System.out.print("  Last Name: " + employee.getLastName
            System.out.println("  Salary: " + employee.getSalary()
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to UPDATE salary for an employee */
   public void updateEmployee(Integer EmployeeID, int salary ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                    (Employee)session.get(Employee.class, EmployeeI
         employee.setSalary( salary );
```

```
         session.update(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to DELETE an employee from the records */
   public void deleteEmployee(Integer EmployeeID){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                   (Employee)session.get(Employee.class, EmployeeID
         session.delete(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
}
```

## 编译和执行：

下面是步骤来编译并运行上述应用程序。请确保已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建hibernate.cfg.xml配置文件中配置章节解释。
- 创建Employee.hbm.xml映射文件，如上图所示。
- 创建Employee.java源文件，如上图所示，并编译它。
- 创建ManageEmployee.java源文件，如上图所示，并编译它。
- 执行ManageEmployee二进制文件来运行程序。

会得到以下结果，并记录将在EMPLOYEE表中创建。

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Zara  Last Name: Ali  Salary: 1000
First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
First Name: Zara  Last Name: Ali  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
```

如果检查EMPLOYEE表，它应该有以下记录：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 29 | Zara       | Ali       |   5000 |
| 31 | John       | Paul      |  10000 |
+----+------------+-----------+--------+
2 rows in set (0.00 sec

mysql>
```

# Hibernate O/R映射 - hibernate

到目前为止，我们已经看到了非常基本的O/R映射使用Hibernate，但也有我们必须学会在细节三个最重要的映射绘制的主题。这些都是集合的映射，关联的实体类和组件映射之间的映射。

## 集合映射：

如果一个实体或类有收集特定变量的值，那么我们就可以用在Java中可用的集合接口中的任何一个对应的值。 Hibernate可以持久java.util.Map，java.util.Set中，java.util.SortedMap，java.util.SortedSet，java.util.List和持久性的实体或值的任意阵列的实例。

| 集合类型 | 映射和说明 |
|---|---|
| java.util.Set | This is mapped with a <set> element and initialized with java.util.HashSet |
| java.util.SortedSet | This is mapped with a <set> element and initialized with java.util.TreeSet. The **sort** attribute can be set to either a comparator or natural ordering. |
| java.util.List | This is mapped with a <list> element and initialized with java.util.ArrayList |
| java.util.Collection | This is mapped with a <bag> or <ibag> element and initialized with java.util.ArrayList |
| java.util.Map | This is mapped with a <map> element and initialized with java.util.HashMap |
| java.util.SortedMap | This is mapped with a <map> element and initialized with java.util.TreeMap. The **sort** attribute can be set to either a comparator or natural ordering. |

数组是由Hibernate与<primitive-array>对Java基本值类型和针对的<array>所有其它支持。然而，他们很少用，所以我不打算讨论这些问题在本教程中。

如果要映射是不直接支持Hibernate的用户定义的集合接口，需要告诉Hibernate有关自定义集合是不太容易的，不建议使用的语义。

## 关联关系映射：

关联实体类和表之间的关系之间的映射是ORM的灵魂。以下是4的方法，使对象之间的关系的基数可以表示。关联映射可以是单向和双向的。

| 映射类型 | 描述 |
|---|---|
| Many-to-One | Mapping many-to-one relationship using Hibernate |
| One-to-One | Mapping one-to-one relationship using Hibernate |
| One-to-Many | Mapping one-to-many relationship using Hibernate |
| Many-to-Many | Mapping many-to-many relationship using Hibernate |

## 组件映射：

这是非常有可能是一个实体类可以有一个引用到另一个类的一个成员变量。如果提到类没有它自己的生命周期，并完全依赖于所属的实体类的生命周期被引用类，因此因此被称为Component类。

收集组件的映射也是可能以类似的方式只是作为与次要配置差异正规集合的映射。我们将看到这两个映射详细的例子。

| 映射类型 | 描述 |
|---|---|
| Component Mappings | 映射一类具有参考到另一个类的一个成员变量。 |

# Hibernate注解 - hibernate

到目前为止，已经看到Hibernate如何使用XML映射文件从POJO的数据到数据库表的改造，反之亦然。 Hibernate注解是一个没有使用XML文件来定义映射的最新方法。可以在除或替换的XML映射元数据使用注解。

Hibernate的注解是强大的方式来提供元数据对象和关系表的映射。所有的元数据被杵到一起的代码POJO java文件这可以帮助用户在开发过程中同时要了解表的结构和POJO。

如果打算让应用程序移植到其他EJB3规范的ORM应用程序，必须使用注解来表示映射信息，但仍然如果想要更大的灵活性，那么应该使用基于XML的映射去。

## 环境设置Hibernate注释

首先，必须确保使用的是JDK5.0，否则，需要JDK升级到JDK5.0带注解的原生支持的优势。

其次，需要安装Hibernate的3.x注解分发包，可从SourceForge上：(Download Hibernate Annotation) 并拷贝 hibernate-annotations.jar, lib/hibernate-comons-annotations.jar 和 lib/ejb3-persistence.jar 从Hibernate注解分配到CLASSPATH

## 注释的类实例：

正如提到的，同时使用Hibernate注释工作的所有元数据杵成随着代码的POJO java文件上面这可以帮助用户在开发过程中同时了解表结构和POJO。

考虑到将要使用下面的EMPLOYEE表来存储的对象：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

下面是用注解来映射与定义的EMPLOYEE表的对象Employee类的映射：

```
import javax.persistence.*;

@Entity
@Table(name = "EMPLOYEE")
public class Employee {
   @Id @GeneratedValue
   @Column(name = "id")
   private int id;

   @Column(name = "first_name")
   private String firstName;

   @Column(name = "last_name")
   private String lastName;

   @Column(name = "salary")
   private int salary;

   public Employee() {}
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

Hibernate检测@Id注释是对一个字段，并假设它应该直接通过在运行时域访问一个对象的属性。如果将@Id注释getId（）方法，将通过getter和setter方法默认情况下允许访问属性。因此，所有其他注释也被放置在任一字段或getter方法，以下所选择的策略。下面的部分将解释在上面的类中使用的注释。

## @Entity 注解:

在EJB3规范说明都包含在javax.persistence包，所以我们导入这个包作为第一步。其次，我们使用了@Entity注解来这标志着这个类作为一个实体bean Employee类，因此它必须有一个无参数的构造函数，总算是有保护的范围可见。

## @Table 注解:

@Table注释允许指定的表将被用于保存该实体在数据库中的详细信息。

@Table注释提供了四个属性，允许覆盖表的名称，它的目录，它的架构，并执行对列的唯一约束在表中。现在，我们使用的是刚刚是EMPLOYEE表的名称。

## @Id 和 @GeneratedValue 注解:

每个实体bean将有一个主键，注释在类的@Id注解。主键可以是单个字段或根据表结构的多个字段的组合。

默认情况下，@Id注解会自动确定要使用的最合适的主键生成策略，但可以通过应用@GeneratedValue注释，它接受两个参数，strategy和generator，不打算在这里讨论，只使用默认的默认键生成策略。让Hibernate确定要使用的generator类型使不同数据库之间代码的可移植性。

## @Column 注解:

@Column批注用于指定的列到一个字段或属性将被映射的细节。可以使用列注释以下最常用的属性：

- name属性允许将显式指定列的名称。
- length 属性允许用于映射一个value尤其是对一个字符串值的列的大小。
- nullable 属性允许该列被标记为NOT NULL生成架构时。
- unique 属性允许被标记为只包含唯一值的列。

## 创建应用程序类 ：

最后，我们将创建应用程序类的main（）方法来运行应用程序。我们将使用这个应用程序，以节省一些员工的记录，然后我们将申请CRUD操作上的记录。

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;
```

```
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try{
         factory = new AnnotationConfiguration().
                   configure().
                   //addPackage("com.xyz") //add package if used.
                   addAnnotatedClass(Employee.class).
                   buildSessionFactory();
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object
         throw new ExceptionInInitializerError(ex);
      }
      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 10000);

      /* List down all the employees */
      ME.listEmployees();

      /* Update employee's records */
      ME.updateEmployee(empID1, 5000);

      /* Delete an employee from the database */
      ME.deleteEmployee(empID2);

      /* List down new list of the employees */
      ME.listEmployees();
   }
   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salar
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         Employee employee = new Employee();
         employee.setFirstName(fname);
         employee.setLastName(lname);
         employee.setSalary(salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      }catch (HibernateException e) {
```

```
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return employeeID;
   }
   /* Method to  READ all the employees */
   public void listEmployees( ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         List employees = session.createQuery("FROM Employee").list
         for (Iterator iterator =
                           employees.iterator(); iterator.hasNext()
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName
            System.out.print("  Last Name: " + employee.getLastName
            System.out.println("  Salary: " + employee.getSalary()
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to UPDATE salary for an employee */
   public void updateEmployee(Integer EmployeeID, int salary ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                    (Employee)session.get(Employee.class, EmployeeI
         employee.setSalary( salary );
         session.update(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to DELETE an employee from the records */
   public void deleteEmployee(Integer EmployeeID){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
```

```
         Employee employee =
                   (Employee)session.get(Employee.class, EmployeeII
         session.delete(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
}
```

## 数据库配置：

现在，让我们创建hibernate.cfg.xml配置文件来定义数据库的相关参数。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
   <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
   </property>
   <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
   </property>

   <!-- Assume students is the database name -->
   <property name="hibernate.connection.url">
      jdbc:mysql://localhost/test
   </property>
   <property name="hibernate.connection.username">
      root
   </property>
   <property name="hibernate.connection.password">
      cohondob
   </property>

</session-factory>
</hibernate-configuration>
```

## 编译和执行：

下面是步骤来编译并运行上述应用程序。请确保已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 从路径中删除Employee.hbm.xml映射文件。
- 创建Employee.java源文件，如上图所示，并编译它。
- 创建ManageEmployee.java源文件，如上图所示，并编译它。
- 执行ManageEmployee二进制文件来运行程序。

会得到以下结果，并记录将在EMPLOYEE表中。

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Zara  Last Name: Ali  Salary: 1000
First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
First Name: Zara  Last Name: Ali  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
```

如果检查EMPLOYEE表，它应该有以下记录：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 29 | Zara       | Ali       |   5000 |
| 31 | John       | Paul      |  10000 |
+----+------------+-----------+--------+
2 rows in set (0.00 sec

mysql>
```

# Hibernate查询语言 - hibernate

Hibernate查询语言（HQL）是一种面向对象的查询语言，类似于SQL，但不是对表和列操作，HQL适用于持久对象和它们的属性。 HQL查询由Hibernate转换成传统的SQL查询，这在圈上的数据库执行操作。

虽然可以直接使用SQL语句和Hibernate使用原生SQL，但建议使用HQL尽可能避免数据库可移植性的麻烦，并采取Hibernate的SQL生成和缓存策略的优势。

都像SELECT，FROM和WHERE等关键字不区分大小写，但如表名和列名的属性是区分在HQL敏感。

## FROM 语句

使用FROM子句，如果要加载一个完整的持久化对象到内存中。下面是一个使用FROM子句的简单的语法：

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

如果需要完全限定在HQL一个类名，只需指定如下的包和类名：

```
String hql = "FROM com.hibernatebook.criteria.Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

## AS 语句

AS子句可以用来别名分配给类中的HQL查询，特别是当有很长的查询。例如，我们前面简单的例子是以下几点：

```
String hql = "FROM Employee AS E";
Query query = session.createQuery(hql);
List results = query.list();
```

AS关键字是可选的，也可以直接在之后的类名指定别名，如下所示：

```
String hql = "FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

# SELECT 子句

SELECT子句提供了更多的控制权比from子句的结果集。如果想获得对象而不是整个对象的几个属性，使用SELECT子句。下面是一个使用SELECT语句来获取Employee对象只是FIRST_NAME字段的简单的语法：

```
String hql = "SELECT E.firstName FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

值得注意的是在这里，Employee.firstName是Employee对象的一个属性，而不是EMPLOYEE表的一个字段。

# WHERE 子句

如果想缩小了从存储返回的特定对象，可以使用WHERE子句。下面是一个使用WHERE子句的简单的语法：

```
String hql = "FROM Employee E WHERE E.id = 10";
Query query = session.createQuery(hql);
List results = query.list();
```

# ORDER BY 子句

若要排序HQL查询的结果，将需要使用ORDER BY子句。您可以在结果集按升序（ASC）或降序（DESC）通过在对象的任何属性排序结果。下面是一个使用ORDER BY子句的简单的语法：

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DES
Query query = session.createQuery(hql);
List results = query.list();
```

如果想通过一个以上的属性进行排序，你会仅仅是额外的属性添加到由子句用逗号隔开，如下所示的命令的结尾：

```
String hql = "FROM Employee E WHERE E.id > 10 " +
             "ORDER BY E.firstName DESC, E.salary DESC ";
Query query = session.createQuery(hql);
List results = query.list();
```

# GROUP BY 子句

该子句允许从Hibernate的它基于属性的值的数据库和组提取信息，并且通常使用结果包括总值。下面是一个使用GROUP BY子句的语法很简单：

```
String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E " +
             "GROUP BY E.firstName";
Query query = session.createQuery(hql);
List results = query.list();
```

# 使用命名参数

Hibernate命名在其HQL查询参数支持。这使得编写接受来自用户的输入容易，不必对SQL注入攻击防御HQL查询。下面是一个使用命名参数的简单的语法：

```
String hql = "FROM Employee E WHERE E.id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id",10);
List results = query.list();
```

# UPDATE 子句

批量更新是新的HQL与Hibernate3，以及不同的删除工作，在Hibernate 3和Hibernate2一样。 Query接口现在包含一个名为executeUpdate()方法用于执行HQL UPDATE或DELETE语句。

在UPDATE子句可以用于更新一个或多个对象中的一个或多个属性。下面是一个使用UPDATE子句的简单的语法：

```
String hql = "UPDATE Employee set salary = :salary "  +
             "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("salary", 1000);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

# DELETE 子句

DELETE子句可以用来删除一个或多个对象。下面是一个使用DELETE子句的简单的语法：

```
String hql = "DELETE FROM Employee "  +
             "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

# INSERT 子句

HQL支持INSERT INTO子句中只记录在那里可以插入从一个对象到另一个对象。以下是使用INSERT INTO子句的简单的语法：

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)"  -
             "SELECT firstName, lastName, salary FROM old_employee'
Query query = session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

# 聚合方法

HQL支持多种聚合方法，类似于SQL。他们工作在HQL同样的方式在SQL和下面的可用功能列表：

| S.N. | 方法 | 描述 |
|---|---|---|
| 1 | avg(property name) | The average of a property's value |
| 2 | count(property name or *) | The number of times a property occurs in the results |
| 3 | max(property name) | The maximum value of the property values |
| 4 | min(property name) | The minimum value of the property values |
| 5 | sum(property name) | The sum total of the property values |

DISTINCT关键字只计算在该行设定的唯一值。下面的查询将只返回唯一的计数：

```
String hql = "SELECT count(distinct E.firstName) FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

## 使用查询分页

有用于分页查询接口的两个方法。

| S.N. | 方法与说明 |
|---|---|
| 1 | **Query setFirstResult(int startPosition)** This method takes an integer that represents the first row in your result set, starting with row 0. |
| 2 | **Query setMaxResults(int maxResult)** This method tells Hibernate to retrieve a fixed number **maxResults** of objects. |

采用上述两种方法一起，可以在网站或Swing应用程序构建一个分页组件。下面是例子，可以扩展来获取10行：

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
query.setFirstResult(1);
query.setMaxResults(10);
List results = query.list();
```

# Hibernate查询条件 - hibernate

Hibernate提供了操作对象，并依次数据在RDBMS表可用的备用方式。其中一个方法是标准的API，它允许你建立一个标准的查询对象编程，可以套用过滤规则和逻辑条件。

Hibernate的Session接口提供了可用于创建一个返回的持久化对象的类的实例时，应用程序执行一个条件查询一个Criteria对象createCriteria()方法。

以下是最简单的一个条件查询的例子是将简单地返回对应于Employee类的每个对象。

```
Criteria cr = session.createCriteria(Employee.class);
List results = cr.list();
```

## 限制与标准：

可以使用add（）方法可用于Criteria对象添加限制条件查询。下面是例子增加一个限制与薪水返回的记录是等于2000：

```
Criteria cr = session.createCriteria(Employee.class);
cr.add(Restrictions.eq("salary", 2000));
List results = cr.list();
```

以下是几个例子覆盖不同的场景，并且可以根据要求使用：

```
Criteria cr = session.createCriteria(Employee.class);

// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To get records having salary less than 2000
cr.add(Restrictions.lt("salary", 2000));

// To get records having fistName starting with zara
cr.add(Restrictions.like("firstName", "zara%"));

// Case sensitive form of the above restriction.
cr.add(Restrictions.ilike("firstName", "zara%"));

// To get records having salary in between 1000 and 2000
cr.add(Restrictions.between("salary", 1000, 2000));

// To check if the given property is null
cr.add(Restrictions.isNull("salary"));

// To check if the given property is not null
cr.add(Restrictions.isNotNull("salary"));

// To check if the given property is empty
cr.add(Restrictions.isEmpty("salary"));

// To check if the given property is not empty
cr.add(Restrictions.isNotEmpty("salary"));
```

可以创建AND或OR使用LogicalExpression限制如下条件：

```
Criteria cr = session.createCriteria(Employee.class);

Criterion salary = Restrictions.gt("salary", 2000);
Criterion name = Restrictions.ilike("firstNname","zara%");

// To get records matching with OR condistions
LogicalExpression orExp = Restrictions.or(salary, name);
cr.add( orExp );

// To get records matching with AND condistions
LogicalExpression andExp = Restrictions.and(salary, name);
cr.add( andExp );

List results = cr.list();
```

虽然上述所有条件，可以直接使用HQL在前面的教程中介绍。

# 分页使用标准：

还有的标准接口，用于分页的两种方法。

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **public Criteria setFirstResult(int firstResult)** This method takes an integer that represents the first row in your result set, starting with row 0. |
| 2 | **public Criteria setMaxResults(int maxResults)** This method tells Hibernate to retrieve a fixed number **maxResults** of objects. |

采用上述两种方法一起，我们可以在我们的网站或Swing应用程序构建一个分页组件。下面是例子，可以扩展来每次获取10行：

```
Criteria cr = session.createCriteria(Employee.class);
cr.setFirstResult(1);
cr.setMaxResults(10);
List results = cr.list();
```

# 排序的结果：

标准的API提供了org.hibernate.criterion.Order类排序按升序或降序排列你的结果集，根据对象的属性。这个例子演示了如何使用Order类的结果集进行排序：

```
Criteria cr = session.createCriteria(Employee.class);
// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To sort records in descening order
crit.addOrder(Order.desc("salary"));

// To sort records in ascending order
crit.addOrder(Order.asc("salary"));

List results = cr.list();
```

# 预测与聚合：

该Criteria API提供了一个org.hibernate.criterion.Projections类可用于获取平均值，最大值或最小值的属性值。Projections类是类似于类限制，因为它提供了几个静态工厂方法用于获得Projection 实例。 provides the

以下是涉及不同的方案的一些例子，可按规定使用：

```
Criteria cr = session.createCriteria(Employee.class);

// To get total row count.
cr.setProjection(Projections.rowCount());

// To get average of a property.
cr.setProjection(Projections.avg("salary"));

// To get distinct count of a property.
cr.setProjection(Projections.countDistinct("firstName"));

// To get maximum of a property.
cr.setProjection(Projections.max("salary"));

// To get minimum of a property.
cr.setProjection(Projections.min("salary"));

// To get sum of a property.
cr.setProjection(Projections.sum("salary"));
```

## Criteria Queries 例子：

考虑下面的POJO类：

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

让我们创建下面的EMPLOYEE表来存储Employee对象：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

以下将被映射文件。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string",
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

最后，我们将创建应用程序类的main（）方法来运行，我们将使用Criteria查询的应用程序：

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.Criteria;
import org.hibernate.criterion.Restrictions;
import org.hibernate.criterion.Projections;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try{
         factory = new Configuration().configure().buildSessionFact
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object
         throw new ExceptionInInitializerError(ex);
      }
      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 2000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 5000);
```

```
      Integer empID4 = ME.addEmployee("Mohd", "Yasee", 3000);

      /* List down all the employees */
      ME.listEmployees();

      /* Print Total employee's count */
      ME.countEmployee();

      /* Print Toatl salary */
      ME.totalSalary();
   }
   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salar
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return employeeID;
   }

   /* Method to  READ all the employees having salary more than 200
   public void listEmployees( ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);
         // Add restriction.
         cr.add(Restrictions.gt("salary", 2000));
         List employees = cr.list();

         for (Iterator iterator =
                           employees.iterator(); iterator.hasNext()
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName
            System.out.print("  Last Name: " + employee.getLastName
            System.out.println("  Salary: " + employee.getSalary()
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
```

```
         session.close();
      }
   }
   /* Method to print total number of records */
   public void countEmployee(){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);

         // To get total row count.
         cr.setProjection(Projections.rowCount());
         List rowCount = cr.list();

         System.out.println("Total Coint: " + rowCount.get(0) );
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
  /* Method to print sum of salaries */
   public void totalSalary(){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);

         // To get total salary.
         cr.setProjection(Projections.sum("salary"));
         List totalSalary = cr.list();

         System.out.println("Total Salary: " + totalSalary.get(0) 
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
}
```

## 编译和执行：

下面是步骤来编译并运行上述应用程序。请确保您已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建hibernate.cfg.xml配置文件中配置章节解释。
- 创建Employee.hbm.xml映射文件，如上图所示。
- 创建Employee.java源文件，如上图所示，并编译它。
- 创建ManageEmployee.java源文件，如上图所示，并编译它。
- 执行ManageEmployee二进制运行程序.

会得到以下结果，并记录将创建在EMPLOYEE表中。

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 5000
First Name: Mohd  Last Name: Yasee  Salary: 3000
Total Coint: 4
Total Salary: 15000
```

如果检查EMPLOYEE表，它应该记录如下：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 14 | Zara       | Ali       |   2000 |
| 15 | Daisy      | Das       |   5000 |
| 16 | John       | Paul      |   5000 |
| 17 | Mohd       | Yasee     |   3000 |
+----+------------+-----------+--------+
4 rows in set (0.00 sec)

mysql>
```

# Hibernate原生SQL - hibernate

可以使用原生SQL来表达数据库查询，如果想利用数据库特有的功能，如查询提示或者Oracle中的CONNECT关键字。 Hibernate3.x允许使用手写SQL语句，包括存储过程，所有的创建，更新，删除和load操作。

应用程序将从会话创建一个原生SQL查询（Session接口上）createSQLQuery()方法：

```
public SQLQuery createSQLQuery(String sqlString) throws HibernateE>
```

当传递一个包含SQL查询到createSQLQuery()方法，可以将SQL结果与任何现有的Hibernate实体，联接，或者一个标量结果使用addEntity()方法，addJoin()，和addScalar()方法关联的字符串。

## 标量查询：

最基本的SQL查询是从一个或多个表中得到标量（数值）的列表。以下是语法使用原生SQL标量的值：

```
String sql = "SELECT first_name, salary FROM EMPLOYEE";
SQLQuery query = session.createSQLQuery(sql);
query.setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP);
List results = query.list();
```

## 实体的查询：

上面的查询都是返回标量值，也就是从resultset中返回的“裸”数据。以下是语法通过addEntity（）方法来从原生SQL查询获得实体对象作为一个整体。

```
String sql = "SELECT * FROM EMPLOYEE";
SQLQuery query = session.createSQLQuery(sql);
query.addEntity(Employee.class);
List results = query.list();
```

## 命名SQL查询：

以下是语法通过addEntity（）方法来从原生SQL查询获得实体对象和使用命名SQL查询。

```
String sql = "SELECT * FROM EMPLOYEE WHERE id = :employee_id";
SQLQuery query = session.createSQLQuery(sql);
query.addEntity(Employee.class);
query.setParameter("employee_id", 10);
List results = query.list();
```

## Native SQL 例子：

考虑下面的POJO类：

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

让我们创建下面的EMPLOYEE表来存储Employee对象：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

以下将被映射文件。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

最后，我们将创建应用程序类的main（）方法来运行，我们将使用原生SQL查询的应用程序：

```
import java.util.*;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.SQLQuery;
import org.hibernate.Criteria;
import org.hibernate.Hibernate;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
```

```
      try{
         factory = new Configuration().configure().buildSessionFact
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object
         throw new ExceptionInInitializerError(ex);
      }
      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 2000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 5000);
      Integer empID4 = ME.addEmployee("Mohd", "Yasee", 3000);

      /* List down employees and their salary using Scalar Query */
      ME.listEmployeesScalar();

      /* List down complete employees information using Entity Quer
      ME.listEmployeesEntity();
   }
   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salar
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return employeeID;
   }

   /* Method to  READ all the employees using Scalar Query */
   public void listEmployeesScalar( ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         String sql = "SELECT first_name, salary FROM EMPLOYEE";
         SQLQuery query = session.createSQLQuery(sql);
         query.setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP);
         List data = query.list();

         for(Object object : data)
         {
            Map row = (Map)object;
```

```
            System.out.print("First Name: " + row.get("first_name")
            System.out.println(", Salary: " + row.get("salary"));
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }

   /* Method to  READ all the employees using Entity Query */
   public void listEmployeesEntity( ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         String sql = "SELECT * FROM EMPLOYEE";
         SQLQuery query = session.createSQLQuery(sql);
         query.addEntity(Employee.class);
         List employees = query.list();

         for (Iterator iterator =
                           employees.iterator(); iterator.hasNext()
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName
            System.out.print("  Last Name: " + employee.getLastName
            System.out.println("  Salary: " + employee.getSalary()
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
}
```

# 编译和执行：

下面是步骤来编译并运行上述应用程序。请确保在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建hibernate.cfg.xml配置文件中配置章节解释。
- 创建Employee.hbm.xml映射文件，如上图所示。
- 创建Employee.java源文件，如上图所示，并编译它。

- 创建ManageEmployee.java源文件，如上图所示，并编译它。
- 执行ManageEmployee二进制文件来运行程序。

会得到以下结果，并记录将在EMPLOYEE表中创建。

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Zara, Salary: 2000
First Name: Daisy, Salary: 5000
First Name: John, Salary: 5000
First Name: Mohd, Salary: 3000
First Name: Zara  Last Name: Ali  Salary: 2000
First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 5000
First Name: Mohd  Last Name: Yasee  Salary: 3000
```

如果检查EMPLOYEE表，它应该记录下已：

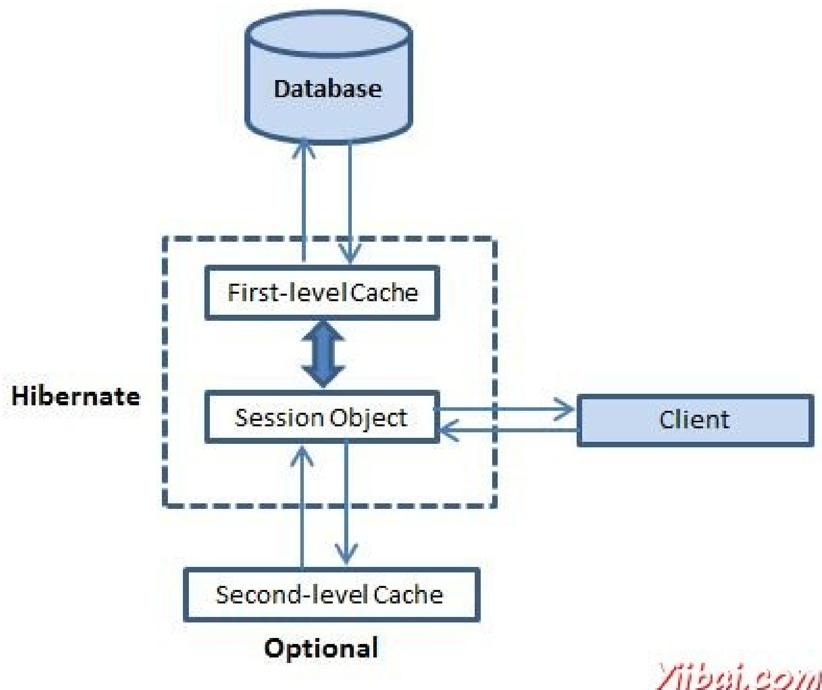
```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 26 | Zara       | Ali       |   2000 |
| 27 | Daisy      | Das       |   5000 |
| 28 | John       | Paul      |   5000 |
| 29 | Mohd       | Yasee     |   3000 |
+----+------------+-----------+--------+
4 rows in set (0.00 sec)

mysql>
```

# Hibernate缓存 - hibernate

缓存是所有关于应用程序的性能优化和它位于应用程序和数据库之间，以避免数据库访问多次，让性能关键型应用程序有更好的表现。

缓存对Hibernate很重要，它采用了多级缓存方案下文所述：




## 第一级缓存：

第一级缓存是Session的缓存，是一个强制性的缓存，通过它所有的请求都必须通过。 Session对象不断自身的动力的对象，提交到数据库之前。

如果发出多个更新一个对象，Hibernate试图拖延尽可能长的时间做了更新，以减少发出的更新SQL语句的数量。如果您关闭会话，所有被缓存的对象都将丢失，要么持久，或在数据库中更新。

## 二级缓存：

二级缓存是可选的缓存和一级缓存，总是会征询任何试图找到一个对象的二级缓存之前。第二级缓存可以在每个类和每个集合基础上进行配置，主要负责在会话缓存的对象。

任何第三方缓存可以使用Hibernate。org.hibernate.cache.CacheProvider接口提供，必须实施提供Hibernate一个句柄缓存实现。

## 查询级别缓存：

Hibernate也实现了查询结果集缓存与二级缓存的紧密集成在一起。

这是一个可选功能，需要两个额外的物理缓存中保存缓存的查询结果和地区当一个表的最后更新的时间戳。这只是针对那些使用相同的参数经常运行的查询非常有用。

## 二级缓存：

Hibernate使用一级缓存，默认情况下，你什么都没有做使用第一级缓存。让我们直接进入可选的第二级缓存。并不是所有的类受益于缓存，这样一来就能禁用二级缓存是很重要的

Hibernate二级缓存被设置为两个步骤。首先，必须决定要使用的并发策略。在此之后，可以配置缓存过期和使用缓存提供物理缓存属性。

## 并发策略：

并发策略是一个中介的负责存储数据项在缓存并从缓存中检索它们。如果要启用二级缓存，将必须决定，为每个持久化类和集合，要使用的缓存并发策略。

- Transactional: 使用这种策略的主要读取数据的地方，以防止过时的数据的并发事务，在更新的罕见情况下是至关重要的。
- Read-write: 再次使用这种策略的主要读取数据的地方，以防止并发事务陈旧的数据是至关重要的，在更新的罕见情况。
- Nonstrict-read-write: 这种策略不保证缓存与数据库之间的一致性。使用此策略，如果数据很少改变和陈旧数据的可能性很小关键是不关注。
- Read-only: 并发策略适用于数据，永远不会改变。使用数据仅供参考。

如果我们要使用第二级缓存为我们的Employee类，让我们添加告诉Hibernate使用可读写的高速缓存策略Employee实例所需的映射元素。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <cache usage="read-write"/>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string",
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

usage="read-write" 属性告诉Hibernate使用一个可读写的并发策略定义的缓存。

## 缓存提供者：

考虑到会用你的缓存候选类的并发策略后，下一步就是选择一个缓存提供程序。Hibernate迫使选择一个缓存提供整个应用程序。

| S.N. | 缓存名称 | 描述 |
|---|---|---|
| 1 | EHCache | 它可以在内存或磁盘上以及群集缓存缓存，它支持可选的Hibernate查询结果缓存。 |
| 2 | OSCache | 支持缓存内存和磁盘在一个JVM中，有一组丰富的过期策略和查询缓存的支持。 |
| 3 | warmCache | 基于JGroups的集群缓存。它使用群集失效，但不支持Hibernate的查询缓存 |
| 4 | JBoss Cache | 一个完全的事务复制的集群缓存也是基于JGroups的组播库。它支持复制或失效，同步或异步通信，乐观和悲观锁定。支持Hibernate的查询缓存 |

每一个缓存提供程序是不是与每个并发策略兼容。以下兼容性矩阵将帮助选择合适的组合。

| Strategy/Provider | Read-only | Nonstrictread-write | Read-write | Transactional |
|---|---|---|---|---|
| EHCache | X | X | X | |
| OSCache | X | X | X | |
| SwarmCache | X | X | | |
| JBoss Cache | X | X | | |

在指定hibernate.cfg.xml配置文件中的缓存提供。选择EHCache作为第二级缓存提供程序：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
   <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
   </property>
   <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
   </property>

   <!-- Assume students is the database name -->
   <property name="hibernate.connection.url">
      jdbc:mysql://localhost/test
   </property>
   <property name="hibernate.connection.username">
      root
   </property>
   <property name="hibernate.connection.password">
      root123
   </property>
   <property name="hibernate.cache.provider_class">
      org.hibernate.cache.EhCacheProvider
   </property>

   <!-- List of XML mapping files -->
   <mapping resource="Employee.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

现在，需要指定缓存区域的属性。EHCache都有自己的配置文件ehcache.xml，在应用程序在CLASSPATH中。在ehcache.xml中Employee类高速缓存配置可能看起来像这样：

```
<diskStore path="java.io.tmpdir"/>
<defaultCache
maxElementsInMemory="1000"
eternal="false"
timeToIdleSeconds="120"
timeToLiveSeconds="120"
overflowToDisk="true"
/>

<cache name="Employee"
maxElementsInMemory="500"
eternal="true"
timeToIdleSeconds="0"
timeToLiveSeconds="0"
overflowToDisk="false"
/>
```

就这样，现在启用Employee类的二级缓存和Hibernate现在二级缓存，每当浏览到一个雇员或当通过标识符加载雇员。

应该分析你所有的类，并选择适当的缓存策略为每个类。有时，二级缓存可能降级的应用程序的性能。所以建议到基准应用程序第一次没有启用缓存，非常适合缓存和检查性能。如果缓存不提高系统性能再有就是在使任何类型的缓存是没有意义的。

## 查询级别缓存：

使用查询缓存，必须先使用 hibernate.cache.use_query_cache="true"属性配置文件中激活它。如果将此属性设置为true，让Hibernate的在内存中创建所需的高速缓存来保存查询和标识符集。

接下来，使用查询缓存，可以使用Query类的setCacheable（Boolean）方法。例如：

```
Session session = SessionFactory.openSession();
Query query = session.createQuery("FROM EMPLOYEE");
query.setCacheable(true);
List users = query.list();
SessionFactory.closeSession();
```

Hibernate也支持通过一个缓存区域的概念非常细粒度的缓存支持。缓存区是这是给定一个名称缓存的一部分。

```
Session session = SessionFactory.openSession();
Query query = session.createQuery("FROM EMPLOYEE");
query.setCacheable(true);
query.setCacheRegion("employee");
List users = query.list();
SessionFactory.closeSession();
```

此代码使用方法告诉Hibernate来存储和查找在缓存中的员工方面的查询。

# Hibernate批量处理 - hibernate

考虑这样一种情况，当需要使用Hibernate上传大量的记录到数据库中。以下是代码片段实现这一使用Hibernate：

```
Session session = SessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Employee employee = new Employee(.....);
    session.save(employee);
}
tx.commit();
session.close();
```

因为默认情况下，Hibernate会缓存所有的持久对象在session级别的缓存，并最终应用程序会失败并发生OutOfMemoryException某处50,000条记录左右。如果使用的是批量处理与Hibernate解决这个问题。

要使用批量处理功能，首先设置hibernate.jdbc.batch_size为批量大小若干无论是在20或50根据对象的大小。这将告诉每X行插入批次hibernate的容器。为了实现这个在代码中，我们需要做一点修改如下：

```
Session session = SessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Employee employee = new Employee(.....);
    session.save(employee);
    if( i % 50 == 0 ) { // Same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

上面的代码将正常工作INSERT操作，但如果愿意做UPDATE操作，那么可以使用下面的代码实现：

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

ScrollableResults employeeCursor = session.createQuery("FROM EMPLOY
                                    .scroll();
int count = 0;

while ( employeeCursor.next() ) {
   Employee employee = (Employee) employeeCursor.get(0);
   employee.updateEmployee();
   seession.update(employee);
   if ( ++count % 50 == 0 ) {
      session.flush();
      session.clear();
   }
}
tx.commit();
session.close();
```

## 批处理示例：

让我们修改配置文件作为补充hibernate.jdbc.batch_size属性：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
   <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
   </property>
   <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
   </property>

   <!-- Assume students is the database name -->
   <property name="hibernate.connection.url">
      jdbc:mysql://localhost/test
   </property>
   <property name="hibernate.connection.username">
      root
   </property>
   <property name="hibernate.connection.password">
      root123
   </property>
   <property name="hibernate.jdbc.batch_size">
      50
   </property>

   <!-- List of XML mapping files -->
   <mapping resource="Employee.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

考虑下面的POJO Employee类：

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

让我们创建下面的EMPLOYEE表来存储Employee对象：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

下面将映射文件映射员工EMPLOYEE表的对象。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

最后，我们将创建与应用程序类main()方法来运行，我们将使用flush() 和 clear()可Session对象方法，使Hibernate的继续写这些记录到数据库中，而不是它们缓存中的应用内存。

```
import java.util.*;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try{
         factory = new Configuration().configure().buildSessionFact
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object
         throw new ExceptionInInitializerError(ex);
      }
      ManageEmployee ME = new ManageEmployee();

      /* Add employee records in batches */
      ME.addEmployees( );
   }
   /* Method to create employee records in batches */
   public void addEmployees( ){
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         for ( int i=0; i<100000; i++ ) {
            String fname = "First Name " + i;
            String lname = "Last Name " + i;
            Integer salary = i;
            Employee employee = new Employee(fname, lname, salary);
            session.save(employee);
             if( i % 50 == 0 ) {
               session.flush();
               session.clear();
            }
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return ;
   }
}
```

# 编译和执行：

下面是步骤来编译并运行上述应用程序。请确保您已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建hibernate.cfg.xml配置文件，如上面所述。
- 创建Employee.hbm.xml映射文件，如上图所示。
- 创建Employee.java源文件，如上图所示，并编译它。
- 创建ManageEmployee.java源文件，如上图所示，并编译它。
- 执行ManageEmployee二进制运行将在EMPLOYEE表中创建100000条记录程序。

# Hibernate拦截器 - hibernate

正如前面已经了解到，在Hibernate中，对象将被创建并保存。一旦对象已经改变，它必须被保存到数据库中。这个过程一直持续的对象是需要的，直到下一次，并且它将被从持久存储加载。

因此，一个对象通过不同阶段的生命周期和拦截器接口提供了可以在不同的阶段被调用来执行一些任务所需的方法。这些方法是从session到应用程序的回调，允许它被保存在申请前检查和/或操纵持久化对象的属性，更新，删除或加载。以下是拦截器界面中所有可用的方法列表：

| S.N. | Method and 描述 |
|---|---|
| 1 | **findDirty()** This method is be called when the **flush()** method is called on a Session object. |
| 2 | **instantiate()** This method is called when a persisted class is instantiated. |
| 3 | **isUnsaved()** This method is called when an object is passed to the **saveOrUpdate()** method/ |
| 4 | **onDelete()** This method is called before an object is deleted. |
| 5 | **onFlushDirty()** This method is called when Hibernate detects that an object is dirty (ie. have been changed) during a flush i.e. update operation. |
| 6 | **onLoad()** This method is called before an object is initialized. |
| 7 | **onSave()** This method is called before an object is saved. |
| 8 | **postFlush()** This method is called after a flush has occurred and an object has been updated in memory. |
| 9 | **preFlush()** This method is called before a flush. |

Hibernate拦截器为我们提供了完全控制对象如何将目光转向应用程序和数据库。

## 如何使用拦截器？

为了建立一个拦截器可以直接实现Interceptor接口的类或继承自EmptyInterceptor类。以下将简单的步骤来使用Hibernate拦截功能。

## 创建拦截器：

我们将在例子在创建和更新Employee对象时，拦截器的方法将被自动调用继承自EmptyInterceptor。可以实现更多的方法按您的要求。

```
import java.io.Serializable;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.EmptyInterceptor;
import org.hibernate.Transaction;
import org.hibernate.type.Type;

public class MyInterceptor extends EmptyInterceptor {
   private int updates;
   private int creates;
   private int loads;

   public void onDelete(Object entity,
                     Serializable id,
                     Object[] state,
                     String[] propertyNames,
                     Type[] types) {
       // do nothing
   }

   // This method is called when Employee object gets updated.
   public boolean onFlushDirty(Object entity,
                     Serializable id,
                     Object[] currentState,
                     Object[] previousState,
                     String[] propertyNames,
                     Type[] types) {
       if ( entity instanceof Employee ) {
          System.out.println("Update Operation");
          return true;
       }
       return false;
   }
   public boolean onLoad(Object entity,
                    Serializable id,
                    Object[] state,
                    String[] propertyNames,
                    Type[] types) {
       // do nothing
       return true;
   }
   // This method is called when Employee object gets created.
   public boolean onSave(Object entity,
                    Serializable id,
                    Object[] state,
                    String[] propertyNames,
                    Type[] types) {
       if ( entity instanceof Employee ) {
```

```
            System.out.println("Create Operation");
            return true;
         }
         return false;
     }
     //called before commit into database
     public void preFlush(Iterator iterator) {
        System.out.println("preFlush");
     }
     //called after committed into database
     public void postFlush(Iterator iterator) {
        System.out.println("postFlush");
     }
  }
```

## 创建POJO类：

现在，让我们修改一点点，我们使用的EMPLOYEE表和Employee类一起看一个例子：

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

## 创建数据库表：

第二步是在数据库中创建表。会有一个表对应于每一个对象，愿意提供持久性。考虑上述目的需要存储和检索到下面的RDBMS表：

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

## 创建映射配置文件：

这一步是创建一个指示Hibernate如何定义的一个或多个类映射到数据库表的映射文件。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

## 创建应用程序类：

最后，我们将创建应用程序类的main()方法来运行应用程序。这里应该指出的是，在创建会话对象，我们使用了拦截器类作为参数。

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
```

```
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try{
         factory = new Configuration().configure().buildSessionFact
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object
         throw new ExceptionInInitializerError(ex);
      }

      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 10000);

      /* List down all the employees */
      ME.listEmployees();

      /* Update employee's records */
      ME.updateEmployee(empID1, 5000);

      /* Delete an employee from the database */
      ME.deleteEmployee(empID2);

      /* List down new list of the employees */
      ME.listEmployees();
   }
   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salar
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return employeeID;
   }
   /* Method to  READ all the employees */
   public void listEmployees( ){
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
```

```
      try{
         tx = session.beginTransaction();
         List employees = session.createQuery("FROM Employee").list
         for (Iterator iterator =
                           employees.iterator(); iterator.hasNext()
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName
            System.out.print("  Last Name: " + employee.getLastName
            System.out.println("  Salary: " + employee.getSalary()
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to UPDATE salary for an employee */
   public void updateEmployee(Integer EmployeeID, int salary ){
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                    (Employee)session.get(Employee.class, EmployeeI
         employee.setSalary( salary );
         session.update(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to DELETE an employee from the records */
   public void deleteEmployee(Integer EmployeeID){
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                   (Employee)session.get(Employee.class, EmployeeID
         session.delete(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
```

```
}
```

## 编译和执行：

下面是步骤来编译并运行上述应用程序。请确保已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建hibernate.cfg.xml配置文件中配置章节解释。
- 创建Employee.hbm.xml映射文件，如上图所示。
- 创建Employee.java源文件，如上图所示，并编译它。
- 创建MyInterceptor.java源文件，如上图所示，并编译它。
- 创建ManageEmployee.java源文件，如上图所示，并编译它。
- 执行ManageEmployee二进制文件来运行程序。

会得到以下结果，并记录将在EMPLOYEE表中创建。

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

Create Operation
preFlush
postFlush
Create Operation
preFlush
postFlush
Create Operation
preFlush
postFlush
First Name: Zara  Last Name: Ali  Salary: 1000
First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
preFlush
postFlush
preFlush
Update Operation
postFlush
preFlush
postFlush
First Name: Zara  Last Name: Ali  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
preFlush
postFlush
```

如果检查EMPLOYEE表，它应该记录信息下：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 29 | Zara       | Ali       |   5000 |
| 31 | John       | Paul      |  10000 |
+----+------------+-----------+--------+
2 rows in set (0.00 sec

mysql>
```

# iBATIS 教程




iBATIS是一个持久层框架，它能够自动在Java、.NET和Ruby on Rails中的对象和在SQL数据库之间映射， iBATIS更容易更快速，更少的代码构建更好的数据库应用导向。

## 读者

本教程是为Java程序员设计的一个需要了解iBATIS框架，同它的体系结构和实际使用情况。

## 必要条件

在继续本教程之前，您应该对Java编程语言有一个很好的理解。因为要处理SQL映射，所以它需要有对SQL和数据库的概念很好的理解。

# iBATIS介绍，iBATIS是什么？ - ibatis

iBATIS的是一个持久层框架，它能够自动在 Java, .NET, 和Ruby on Rails中与SQL数据库和对象之间的映射。映射是从应用程序逻辑封装在XML配置文件中的SQL语句脱钩。

iBATIS是一个轻量级的框架和持久性API适合持久化的POJO（普通Java对象）。

iBATIS是被称为一个数据映射和映射需要的类的属性和数据库中的表的列之间的参数和结果。

iBATIS和其他持久化框架，如Hibernate之间的显著区别在于，iBATIS强调使用SQL，而其他的框架通常使用一个自定义的查询语言，具有Hibernate查询语言（HQL）或Enterprise JavaBeans的查询语言（EJB QL）。

## iBATIS的设计理念：

iBatis提供了以下的设计理念：

- 简单: iBATIS的被广泛认为是可用的最简单的持久化框架之一。
- 快速开发：iBATIS的理念是尽一切可能，以方便超快速开发。
- 可移植性: iBATIS可用于几乎任何语言或平台，如Java，Ruby和C＃，微软.NET实现。
- 独立的接口：iBATIS提供独立于数据库的接口和API，帮助应用程序的其余部分保持独立的任何持久性相关的资源，
- 开源：iBATIS是自由和开放源码软件。

## IBATIS的优点

下面是使用iBATIS的一些优势：

- 支持存储过程：iBATIS的SQL封装以存储过程的形式，使业务逻辑保持在数据库之外，应用程序更易于部署和测试，更便于移植。
- 支持内嵌的SQL：预编译器不是必需的，并有完全访问所有的SQL语句的特性。
- 支持动态SQL: iBATIS特性提供基于参数动态生成SQL查询。
- 支持O / RM：iBATIS支持许多相同的功能作为一个O / RM工具，如延迟加载，连接抓取，缓存，运行时代码生成和继承

# 先决条件：

在开始之前，要确保你了解过程和面向对象编程的基本知识：控制结构，数据结构和变量，类，对象等。

iBATIS使用Java编程语言开发面向数据库应用程序。

要理解Java编程细节可以通过我们的Java教程。

# iBATIS配置环境 - ibatis

在开始使用iBATIS开发之前，必须设置你的环境正常。本教程将指导您用几个步骤来实现的工作环境。

## iBATIS 安装:

这里有几个简单的步骤，需要开展Linux机器上安装iBATIS：

- 下载iBATIS的最新版本 下载iBATIS.
- 解压下载的文件，从包中提取.jar文件并将其保存在相应的lib目录下。
- 在提取 .jar文件适当设置PATH和CLASSPATH变量。

下面是进行Linux机器下载iBATIS的二进制文件的步骤：

```
$ unzip ibatis-2.3.4.726.zip
  inflating: META-INF/MANIFEST.MF
   creating: doc/
   creating: lib/
   creating: simple_example/
   creating: simple_example/com/
   creating: simple_example/com/mydomain/
   creating: simple_example/com/mydomain/data/
   creating: simple_example/com/mydomain/domain/
   creating: src/
  inflating: doc/dev-javadoc.zip
  inflating: doc/user-javadoc.zip
  inflating: jar-dependencies.txt
  inflating: lib/ibatis-2.3.4.726.jar
  inflating: license.txt
  inflating: notice.txt
  inflating: release.txt
$pwd
/var/home/ibatis
$set PATH=$PATH:/var/home/ibatis/
$set CLASSPATH=$CLASSPATH:/var/home/ibatis
                       /lib/ibatis-2.3.4.726.jar
```

## 数据库设置：

使用下面的语法在 MySQL数据库中创建EMPLOYEE表：

```
mysql> CREATE TABLE EMPLOYEE (
            id INT NOT NULL auto_increment,
            first_name VARCHAR(20) default NULL,
            last_name  VARCHAR(20) default NULL,
            salary     INT  default NULL,
            PRIMARY KEY (id)
        );
```

# 创建SqlMapConfig.xml

考虑以下几点：

- 我们将使用JDBC来访问数据库 testdb.
- MySQL的JDBC驱动程序是 "com.mysql.jdbc.Driver".
- 连接URL是 "jdbc:mysql://localhost:3306/testdb".
- 使用的用户名和密码是 "root" and "root".
- SQL语句映射的所有操作将被描述在"Employee.xml".

基于上述假设，我们必须创建一个XML配置文件，nameSqlMapConfig.xml以下内容。这就是需要提供所需的iBatis的所有配置：

这两个文件SqlMapConfig.xml和Employee.xml 存在于类路径。现在，我们将保持Employee.xml文件为空，我们将格式转换的在随后的章节内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig
PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
     <settings useStatementNamespaces="true"/>
     <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
          <property name="JDBC.Driver"
               value="com.mysql.jdbc.Driver"/>
          <property name="JDBC.ConnectionURL"
               value="jdbc:mysql://localhost:3306/testdb"/>
          <property name="JDBC.Username" value="root"/>
          <property name="JDBC.Password" value="root"/>
        </dataSource>
      </transactionManager>
     <sqlMap resource="Employee.xml"/>
</sqlMapConfig>
```

还有其他一些可选的属性，您可以在SqlMapConfig.xml文件中设置：

```
<property name="JDBC.AutoCommit" value="true"/>

<property name="Pool.MaximumActiveConnections" value="10"/>

<property name="Pool.MaximumIdleConnections" value="5"/>

<property name="Pool.MaximumCheckoutTime" value="150000"/>

<property name="Pool.MaximumTimeToWait" value="500"/>

<property name="Pool.PingQuery" value="select 1 from Employee"/>

<property name="Pool.PingEnabled" value="false"/>
```

# iBATIS创建操作 - ibatis

若要使用iBATIS执行的任何CRUD（创建，写入，更新和删除）操作，需要创建一个的POJO（普通Java对象）类对应的表。本课程介绍的对象，将“模式”的数据库表中的行。

POJO类必须实现所有执行所需的操作所需的方法。

我们已经在MySQL下有EMPLOYEE表：

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

## Employee POJO 类 ：

我们会在Employee.java文件中创建Employee类，如下所示：

```
public class Employee {
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  /* Define constructors for the Employee class. */
  public Employee() {}

  public Employee(String fname, String lname, int salary) {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }
} /* End of Employee */
```

可以定义方法来设置表中的各个字段。下一章节将告诉你如何获得各个字段的值。

## Employee.xml 文件:

要定义使用iBATIS SQL映射语句中，我们将使用<insert>标签，这个标签定义中，我们会定义将用于在IbatisInsert.java文件的数据库执行SQL INSERT查询“id”。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">
<insert id="insert" parameterClass="Employee">

   insert into EMPLOYEE(first_name, last_name, salary)
   values (#first_name#, #last_name#, #salary#)

   <selectKey resultClass="int" keyProperty="id">
      select last_insert_id() as id
   </selectKey>

</insert>

</sqlMap>
```

这里parameterClass：可以采取一个值作为字符串，整型，浮点型，double或根据要求任何类的对象。在这个例子中，我们将通过Employee对象作为参数而调用SqlMap类的insert方法。

如果您的数据库表使用IDENTITY，AUTO_INCREMENT或串行列或已定义的SEQUENCE/GENERATOR，可以使用<selectKey>元素在的<insert>语句中使用或返回数据库生成的值。

## IbatisInsert.java 文件:

文件将应用程序级别的逻辑在Employee表中插入记录：

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisInsert{
  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   /* This would insert one record in Employee table. */
   System.out.println("Going to insert record.....");
   Employee em = new Employee("Zara", "Ali", 5000);

   smc.insert("Employee.insert", em);

   System.out.println("Record Inserted Successfully ");

  }
}
```

## 编译和运行：

下面是步骤来编译并运行上述软件。请确保已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisInsert.java如上图所示，并编译它。
- 执行IbatisInsert二进制文件来运行程序。

会得到下面的结果，并创纪录的将在EMPLOYEE表中创建。

```
$java IbatisInsert
Going to insert record.....
Record Inserted Successfully
```

去查看EMPLOYEE表，它应该有如下结果：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
+----+------------+-----------+--------+
1 row in set (0.00 sec)
```

# iBATIS读取操作 - ibatis

上一章展示了如何使用iBATIS执行创建操作表。本章将告诉你如何使用iBATIS来读取表。

我们已经在MySQL下有EMPLOYEE表：

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

此表有如下只有一条记录：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
+----+------------+-----------+--------+
1 row in set (0.00 sec)
```

## Employee POJO 类:

要执行读操作，我们将修改Employee类中Employee.java文件，如下所示：

```
public class Employee {
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  /* Define constructors for the Employee class. */
  public Employee() {}

  public Employee(String fname, String lname, int salary) {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }

  /* Here are the method definitions */
  public int getId() {
    return id;
  }
  public String getFirstName() {
    return first_name;
  }
  public String getLastName() {
    return last_name;
  }
  public int getSalary() {
    return salary;
  }
} /* End of Employee */
```

## Employee.xml 文件:

要定义使用iBATIS SQL映射语句，我们将增加<select>标记在Employee.xml文件，这个标签定义中，我们会定义将用于在IbatisRead.java文件的数据库执行SQL SELECT查询的“id”。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">
<insert id="insert" parameterClass="Employee">
   INSERT INTO EMPLOYEE(first_name, last_name, salary)
   values (#first_name#, #last_name#, #salary#)

   <selectKey resultClass="int" keyProperty="id">
      select last_insert_id() as id
   </selectKey>

</insert>
<select id="getAll" resultClass="Employee">
   SELECT * FROM EMPLOYEE
</select>
</sqlMap>
```

在这里，我们没有使用WHERE子句和SQL SELECT语句。后续章节将演示如何用WHERE和SELECT语句子句，以及如何将值传递到WHERE子句。

## IbatisRead.java 文件:

该文件将应用程序级别的逻辑从雇员Employee表中读出记录：

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisRead{
  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   /* This would read all records from the Employee table. */
   System.out.println("Going to read records.....");
   List <Employee> ems = (List<Employee>)
                        smc.queryForList("Employee.getAll", null);
   Employee em = null;
   for (Employee e : ems) {
      System.out.print("  " + e.getId());
      System.out.print("  " + e.getFirstName());
      System.out.print("  " + e.getLastName());
      System.out.print("  " + e.getSalary());
      em = e;
      System.out.println("");
   }

   System.out.println("Records Read Successfully ");

  }
}
```

## 编译和运行：

下面是步骤来编译并运行上述应用。请确保您已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisRead.java如上图所示，并编译它。
- 执行IbatisRead二进制文件来运行程序。

你会得到下面的结果，并且将记录从EMPLOYEE表中读取。

```
Going to read records.....
  1  Zara  Ali  5000
Record Reads Successfully
```

# iBATIS更新操作 - ibatis

上一章展示了如何使用iBATIS对表进行读取操作。本章将告诉你如何在一个表中使用iBATIS更新记录。

我们已经在MySQL下有EMPLOYEE表：

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

此表有如下只有一条记录：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
+----+------------+-----------+--------+
1 row in set (0.00 sec)
```

## Employee POJO 类：

要执行UDPATE操作，需要修改Employee.java文件，如下所示：

```
public class Employee {
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  /* Define constructors for the Employee class. */
  public Employee() {}

  public Employee(String fname, String lname, int salary) {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }

 /* Here are the required method definitions */
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getFirstName() {
    return first_name;
  }
  public void setFirstName(String fname) {
    this.first_name = fname;
  }
  public String getLastName() {
    return last_name;
  }
  public void setlastName(String lname) {
    this.last_name = lname;
  }
  public int getSalary() {
    return salary;
  }
  public void setSalary(int salary) {
    this.salary = salary;
  }

 } /* End of Employee */
```

## Employee.xml 文件:

要定义使用iBATIS SQL映射语句，我们想补充的<Update>标签Employee.xml文件，这个标签定义中，我们会定义将用于在IbatisUpdate.java文件的数据库执行SQL UPDATE查询的“id”。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">
<insert id="insert" parameterClass="Employee">
   INSERT INTO EMPLOYEE(first_name, last_name, salary)
   values (#first_name#, #last_name#, #salary#)

   <selectKey resultClass="int" keyProperty="id">
      select last_insert_id() as id
   </selectKey>

</insert>

<select id="getAll" resultClass="Employee">
   SELECT * FROM EMPLOYEE
</select>

<update id="update" parameterClass="Employee">
   UPDATE EMPLOYEE
   SET    first_name = #first_name#
   WHERE  id = #id#
</update>
</sqlMap>
```

## IbatisUpdate.java 文件:

文件将应用程序级别的逻辑来更新记录到Employee表：

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisUpdate{
  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   /* This would update one record in Employee table. */
   System.out.println("Going to update record.....");
   Employee rec = new Employee();
   rec.setId(1);
   rec.setFirstName( "Roma");
   smc.update("Employee.update", rec );
   System.out.println("Record updated Successfully ");

   System.out.println("Going to read records.....");
   List <Employee> ems = (List<Employee>)
                  smc.queryForList("Employee.getAll", null);
   Employee em = null;
   for (Employee e : ems) {
      System.out.print("  " + e.getId());
      System.out.print("  " + e.getFirstName());
      System.out.print("  " + e.getLastName());
      System.out.print("  " + e.getSalary());
      em = e;
      System.out.println("");
   }

   System.out.println("Records Read Successfully ");

  }
}
```

# 编译和运行：

下面是步骤来编译并运行上述软件。请确保您已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisUpdate.java如上图所示，并编译它。

- 执行IbatisUpdate二进制文件来运行程序。

得到下面的结果，并创建纪录在EMPLOYEE表进行更新和更高版本相同的记录将从EMPLOYEE表中读出。

```
Going to update record.....
Record updated Successfully
Going to read records.....
  1  Roma  Ali  5000
Records Read Successfully
```

# iBATIS删除操作 - ibatis

本章将教你如何从表中使用iBATIS删除记录。

我们已经在MySQL下有EMPLOYEE表：

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

假设这个表是有两条记录如下：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
|  2 | Roma       | Ali       |   3000 |
+----+------------+-----------+--------+
2 row in set (0.00 sec)
```

## Employee POJO 类：

要执行删除操作，就需要修改Employee.java文件。因此，让我们保持它不变，在上一章我们使用过。

```
public class Employee {
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  /* Define constructors for the Employee class. */
  public Employee() {}

  public Employee(String fname, String lname, int salary) {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }

 /* Here are the required method definitions */
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getFirstName() {
    return first_name;
  }
  public void setFirstName(String fname) {
    this.first_name = fname;
  }
  public String getLastName() {
    return last_name;
  }
  public void setlastName(String lname) {
    this.last_name = lname;
  }
  public int getSalary() {
    return salary;
  }
  public void setSalary(int salary) {
    this.salary = salary;
  }

 } /* End of Employee */
```

## Employee.xml 文件:

要定义使用iBATIS SQL映射语句中，我们将增加<Delete>键标签Employee.xml文件，这个标签定义中，我们会定义将用于在IbatisDelete.java文件执行SQL DELETE查询数据库的“id”。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">
<insert id="insert" parameterClass="Employee">
   INSERT INTO EMPLOYEE(first_name, last_name, salary)
   values (#first_name#, #last_name#, #salary#)

   <selectKey resultClass="int" keyProperty="id">
      select last_insert_id() as id
   </selectKey>

</insert>

<select id="getAll" resultClass="Employee">
   SELECT * FROM EMPLOYEE
</select>

<update id="update" parameterClass="Employee">
   UPDATE EMPLOYEE
   SET    first_name = #first_name#
   WHERE  id = #id#
</update>

<delete id="delete" parameterClass="int">
   DELETE FROM EMPLOYEE
   WHERE  id = #id#
</delete>

</sqlMap>
```

## IbatisDelete.java 文件:

文件将应用程序级别的逻辑从Employee表中删除记录：

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisDelete{
  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   /* This would delete one record in Employee table. */
   System.out.println("Going to delete record.....");
   int id = 1;

   smc.delete("Employee.delete", id );
   System.out.println("Record deleted Successfully ");

   System.out.println("Going to read records.....");
   List <Employee> ems = (List<Employee>)
                        smc.queryForList("Employee.getAll", null);
   Employee em = null;
   for (Employee e : ems) {
      System.out.print("  " + e.getId());
      System.out.print("  " + e.getFirstName());
      System.out.print("  " + e.getLastName());
      System.out.print("  " + e.getSalary());
      em = e;
      System.out.println("");
   }

   System.out.println("Records Read Successfully ");

  }
}
```

## 编译和运行：

下面是步骤来编译并运行上述软件。请确保已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisDelete.java如上图所示，并编译它。

- 执行IbatisDelete二进制文件来运行程序。

得到以下结果，ID=1将在EMPLOYEE表中被删除，并读取EMPLOYEE表中的一条记录。

```
Going to delete record.....
Record deleted Successfully
Going to read records.....
  2  Roma  Ali  3000
Records Read Successfully
```

# iBATIS结果映射 - ibatis

resultMap的元素是在iBATIS的最重要和最强大的元素。您可以通过使用iBATIS的结果映射减少高达90%的JDBC编码，在某些情况下，可以让你做JDBC不支持的事情。

ResultMaps的设计是这样的简单语句不需要明确的结果映射，以及更复杂的报表要求不超过绝对必要说明的关系。

本章将只给你一个简单的介绍iBATIS的结果映射。

我们已经在MySQL下有EMPLOYEE表：

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

此表有如下两条记录：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
|  2 | Roma       | Ali       |   3000 |
+----+------------+-----------+--------+
2 row in set (0.00 sec)
```

## Employee POJO 类:

使用iBATIS的结果映射，需要修改Employee.java文件。因此，让我们保持它，因为它已经在前一章中使用过。

```
public class Employee {
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  /* Define constructors for the Employee class. */
  public Employee() {}

  public Employee(String fname, String lname, int salary) {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }

 /* Here are the required method definitions */
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getFirstName() {
    return first_name;
  }
  public void setFirstName(String fname) {
    this.first_name = fname;
  }
  public String getLastName() {
    return last_name;
  }
  public void setlastName(String lname) {
    this.last_name = lname;
  }
  public int getSalary() {
    return salary;
  }
  public void setSalary(int salary) {
    this.salary = salary;
  }

 } /* End of Employee */
```

## Employee.xml 文件:

在这里，我们将修改Employee.xml文件介绍<resultMap></ resultMap>标记。这个标签就必须在我们<select>标记的resultMap属性运行此结果映射这是需要一个id。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">

<!-- Perform Insert Operation -->
<insert id="insert" parameterClass="Employee">
   INSERT INTO EMPLOYEE(first_name, last_name, salary)
   values (#first_name#, #last_name#, #salary#)

   <selectKey resultClass="int" keyProperty="id">
      select last_insert_id() as id
   </selectKey>

</insert>

<!-- Perform Read Operation -->
<select id="getAll" resultClass="Employee">
   SELECT * FROM EMPLOYEE
</select>

<!-- Perform Update Operation -->
<update id="update" parameterClass="Employee">
   UPDATE EMPLOYEE
   SET    first_name = #first_name#
   WHERE  id = #id#
</update>

<!-- Perform Delete Operation -->
<delete id="delete" parameterClass="int">
   DELETE FROM EMPLOYEE
   WHERE  id = #id#
</delete>

<!-- Using ResultMap -->
 <resultMap id="result" class="Employee">
    <result property="id" column="id"/>
    <result property="first_name" column="first_name"/>
    <result property="last_name" column="last_name"/>
    <result property="salary" column="salary"/>
</resultMap>
<select id="useResultMap" resultMap="result">
         SELECT * FROM EMPLOYEE
         WHERE id=#id#
</select>

</sqlMap>
```

# IbatisResultMap.java 文件:

文件将应用程序级别的逻辑，从使用结果映射Employee表中读取记录：

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisResultMap{
  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   int id = 1;
   System.out.println("Going to read record.....");
   Employee e = (Employee)smc.queryForObject
                ("Employee.useResultMap", id);

   System.out.println("ID:  " + e.getId());
   System.out.println("First Name:  " + e.getFirstName());
   System.out.println("Last Name:  " + e.getLastName());
   System.out.println("Salary:  " + e.getSalary());

   System.out.println("Record read Successfully ");

  }
}
```

## 编译和运行：

下面是步骤来编译并运行上述软件。请确保您已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisResultMap.java如上图所示，并编译它。
- 执行IbatisResultMap二进制文件来运行程序。

会得到下面的结果是对EMPLOYEE表的读操作。

```
Going to read record.....
ID:  1
First Name:  Zara
Last Name:  Ali
Salary:  5000
Record read Successfully
```

# iBATIS存储过程 - ibatis

使用iBATIS配置来调用存储过程。为了理解这一章，首先需要了解我们是如何在MySQL中创建一个存储过程。

在继续对本章学习之前，可以通过MySQL存储过程。

我们已经在MySQL下有EMPLOYEE表：

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

让我们在MySQL数据库中创建以下存储过程。

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `testdb`.`getEmp` $$
CREATE PROCEDURE `testdb`.`getEmp`
   (IN empid INT)
BEGIN
   SELECT * FROM EMPLOYEE
   WHERE ID = empid;
END $$

DELIMITER;
```

考虑EMPLOYEE表是有两条记录如下：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
|  2 | Roma       | Ali       |   3000 |
+----+------------+-----------+--------+
2 row in set (0.00 sec)
```

# Employee POJO 类：

使用存储过程，你就需要修改Employee.java文件。因此，让我们保持它，因为它是在前一章。

```
public class Employee {
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  /* Define constructors for the Employee class. */
  public Employee() {}

  public Employee(String fname, String lname, int salary) {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }

 /* Here are the required method definitions */
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getFirstName() {
    return first_name;
  }
  public void setFirstName(String fname) {
    this.first_name = fname;
  }
  public String getLastName() {
    return last_name;
  }
  public void setlastName(String lname) {
    this.last_name = lname;
  }
  public int getSalary() {
    return salary;
  }
  public void setSalary(int salary) {
    this.salary = salary;
  }

 } /* End of Employee */
```

## Employee.xml 类：

在这里，我们将修改Employee.xml文件介绍<procedure></procedure>和<parameterMap></parameterMap>标记。这里<procedure></procedure>标签将有一个id，我们会用我们的应用程序来调用存储过程。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">

<!-- Perform Insert Operation -->
<insert id="insert" parameterClass="Employee">
   INSERT INTO EMPLOYEE(first_name, last_name, salary)
   values (#first_name#, #last_name#, #salary#)

   <selectKey resultClass="int" keyProperty="id">
      select last_insert_id() as id
   </selectKey>

</insert>

<!-- Perform Read Operation -->
<select id="getAll" resultClass="Employee">
   SELECT * FROM EMPLOYEE
</select>

<!-- Perform Update Operation -->
<update id="update" parameterClass="Employee">
   UPDATE EMPLOYEE
   SET    first_name = #first_name#
   WHERE  id = #id#
</update>

<!-- Perform Delete Operation -->
<delete id="delete" parameterClass="int">
   DELETE FROM EMPLOYEE
   WHERE  id = #id#
</delete>

<!-- To call stored procedure. -->
<procedure id="getEmpInfo" resultClass="Employee"
              parameterMap="getEmpInfoCall">
   { call getEmp( #acctID# ) }
</procedure>
<parameterMap id="getEmpInfoCall" class="map">
   <parameter property="acctID" jdbcType="INT"
   javaType="java.lang.Integer" mode="IN"/>
</parameterMap>

</sqlMap>
```

## IbatisSP.java 文件:

文件将应用程序级别的逻辑读取使用结果映射Employee表员工的姓名name：

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisSP{
  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   int id = 1;
   System.out.println("Going to read employee name.....");
   Employee e = (Employee)smc.queryForObject
                ("Employee.getEmpInfo", id);

   System.out.println("First Name:  " + e.getFirstName());

   System.out.println("Record name Successfully ");

  }
}
```

## 编译和运行：

下面是步骤来编译并运行上述应用程序。请确保您在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisSP.java如上图所示，并编译它。
- 执行IbatisSP二进制文件来运行程序。

得到以下结果：

```
Going to read record.....
ID:  1
First Name:  Zara
Last Name:  Ali
Salary:  5000
Record read Successfully
```

# iBATIS动态SQL - ibatis

使用动态查询是iBatis一个非常强大的功能。有时你已经改变WHERE子句条件的基础上你的参数对象的状态。在这种情况下的iBATIS提供了一组可以映射语句中使用，以提高SQL语句的重用性和灵活性的动态SQL标签。

所有的逻辑是使用一些额外的标签放在：XML文件。下面是一个例子，其中的SELECT语句将努力在两个方面：

- 如果想传递一个ID，然后它会返回所有与该ID的记录，
- 否则，将返回所有雇员ID为NULL的记录。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">
<select id="findByID" resultClass="Employee">
   SELECT * FROM EMPLOYEE
   <dynamic prepend="WHERE ">
      <isNull property="id">
         id IS NULL
      </isNull>
      <isNotNull property="id">
         id = #id#
      </isNotNull>
   </dynamic>
</select>
</sqlMap>
```

可以使用<isNotEmpty>标签如下检查条件。在此条件下将增加，只有当通过属性不为空。

```
..................
<select id="findByID" resultClass="Employee">
   SELECT * FROM EMPLOYEE
   <dynamic prepend="WHERE ">
      <isNotEmpty property="id">
          id = #id#
      </isNotEmpty>
   </dynamic>
</select>
..................
```

如果想查询对id和/或雇员的名字选取。SELECT语句如下：

```
..................
<select id="findByID" resultClass="Employee">
   SELECT * FROM EMPLOYEE
   <dynamic prepend="WHERE ">
      <isNotEmpty prepend="AND" property="id">
           id = #id#
      </isNotEmpty>
      <isNotEmpty prepend="OR" property="first_name">
           first_name = #first_name#
      </isNotEmpty>
   </dynamic>
</select>
..................
```

# 例如：动态SQL

下面的例子将展示如何编写SELECT语句中使用动态SQL。考虑，我们已经在MySQL下有EMPLOYEE表：

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

此表有如下只有一条记录：

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
|  2 | Roma       | Ali       |   3000 |
|  3 | Noha       | Ali       |   7000 |
+----+------------+-----------+--------+
3 row in set (0.00 sec)
```

# Employee POJO 类：

要执行读取操作，让我们在Employee.java文件Employee类，如下所示：

```
public class Employee {
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  /* Define constructors for the Employee class. */
  public Employee() {}

  public Employee(String fname, String lname, int salary) {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }

  /* Here are the method definitions */
  public int getId() {
    return id;
  }
  public String getFirstName() {
    return first_name;
  }
  public String getLastName() {
    return last_name;
  }
  public int getSalary() {
    return salary;
  }
} /* End of Employee */
```

## Employee.xml 文件:

要定义使用iBATIS SQL映射语句，我们将增加在以下文件Employee.xml修改<select>标记和这个标签定义，我们将定义一个“id”，这将被用于IbatisReadDy.java文件上执行动态SQL的SELECT查询数据库。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">
<select id="findByID" resultClass="Employee">
   SELECT * FROM EMPLOYEE
   <dynamic prepend="WHERE ">
      <isNotNull property="id">
         id = #id#
      </isNotNull>
   </dynamic>
</select>
</sqlMap>
```

上面的SELECT语句将努力在两个方面（一）如果想传递一个ID，然后将相应的编号（二）返回的记录，否则将返回所有记录。

## IbatisReadDy.java 文件:

文件将应用程序级别的逻辑从Employee表读出的条件记录：

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisReadDy{
  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd=Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc=SqlMapClientBuilder.buildSqlMapClient(rd);

   /* This would read all records from the Employee table.*/
   System.out.println("Going to read records.....");
   Employee rec = new Employee();
   rec.setId(1);

   List <Employee> ems = (List<Employee>)
                   smc.queryForList("Employee.findByID", rec);
   Employee em = null;
   for (Employee e : ems) {
      System.out.print("  " + e.getId());
      System.out.print("  " + e.getFirstName());
      System.out.print("  " + e.getLastName());
      System.out.print("  " + e.getSalary());
      em = e;
      System.out.println("");
   }

   System.out.println("Records Read Successfully ");

  }
}
```

## 编译和运行：

下面是步骤来编译并运行上述应用。请确保已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisReadDy.java如上图所示，并编译它。
- 执行IbatisReadDy二进制文件来运行程序。

会得到下面的结果，并且将记录从EMPLOYEE表中读取。

```
Going to read records.....
  1  Zara  Ali  5000
Record Reads Successfully
```

试试上面的例子中通过传递空值作为smc.queryForList（“Employee.findByID”，NULL）。

# iBATIS OGNL 表达式

iBATIS的提供了强大的基于OGNL的表达式来消除其他元素。

- if 语句
- choose, when, otherwise 语句
- where 语句
- foreach 语句

# if语句：

最常见的事情在动态SQL是有条件地包括一个where子句的一部分。例如：

```
<select id="findActiveBlogWithTitleLike"
           parameterType="Blog" resultType="Blog">
   SELECT * FROM BLOG
   WHERE state = 'ACTIVE.
   <if test="title != null">
      AND title like #{title}
   </if>
</select>
```

这条语句会提供功能的可选的文本搜索类型。如果没有传递title，那么所有激活的博客将被退回。但是，如果传递一个标题，它会寻找标题以like 给定的条件。

可以包括多个if条件如下：

最常见的事情在动态SQL是有条件地包括一个where子句的一部分。例如：

```
<select id="findActiveBlogWithTitleLike"
           parameterType="Blog" resultType="Blog">
   SELECT * FROM BLOG
   WHERE state = 'ACTIVE.
   <if test="title != null">
      AND title like #{title}
   </if>
   <if test="author != null">
      AND author like #{author}
   </if>
</select>
```

## choose, when, otherwise 语句:

iBATIS提供了一个选择的元素，它类似于Java的switch语句。这有助于选择很多种情况。

下面的例子将只搜索标题上如果提供，那么只有由作者如果已提供。如果没有提供，让我们只返回精选的博客：

```
<select id="findActiveBlogWithTitleLike"
           parameterType="Blog" resultType="Blog">
   SELECT * FROM BLOG
   WHERE state = 'ACTIVE.
   <choose>
      <when test="title != null">
         AND title like #{title}
      </when>
      <when test="author != null and author.name != null">
         AND author like #{author}
      </when>
      <otherwise>
         AND featured = 1
      </otherwise>
   </choose>
</select>
```

## where 语句:

如果我们看一下前面的例子中，如果没有一个条件满足会发生什么事？最终SQL看起来像这样：

```
SELECT * FROM BLOG
WHERE
```

这会失败，但iBATIS有一个简单的改变一个简单的解决方案，让一切工作正常：

```
<select id="findActiveBlogLike"
           parameterType="Blog" resultType="Blog">
   SELECT * FROM BLOG
   <where>
      <if test="state != null">
         state = #{state}
      </if>
      <if test="title != null">
         AND title like #{title}
      </if>
      <if test="author != null>
         AND author like #{author}
      </if>
   </where>
</select>
```

where元素知道只有插入WHERE ，如果有一个由含标签返回的任何内容。此外，如果该内容开头AND或OR，它知道剥离其关闭。

## foreach语句：

foreach元素是非常强大的，并允许你指定一个集合，声明可以在元素的体内可用于项目和索引变量。

它也允许你指定打开和关闭的字符串，并添加一个分隔符放置在迭代之间。可以建立一个IN条件如下：

```
<select id="selectPostIn" resultType="domain.blog.Post">
   SELECT *
   FROM POST P
   WHERE ID in
   <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
         #{item}
   </foreach>
</select>
```

# iBATIS调试 - ibatis

这是很容易，同时与iBATIS的工作程序进行调试。 iBATIS有内置的日志支持，并适用于下列日志库，并在这个顺序搜索他们。

- Jakarta Commons日志记录（JCL）。
- Log4J
- JDK 日志

可以使用任何上面列出的库在iBATIS。

## 调试和Log4J：

假设你要使用Log4J，这是最好用的日志记录。继续操作之前，需要交叉检查以下几点：

- Log4J JAR 文件 (log4j-{version}.jar) 应在CLASSPATH中。
- 必须在CLASSPATH中提供log4j.properties。

下面是一个log4j.properties文件。请注意，某些行被注释掉了。你可以取消他们，如果你需要额外的调试信息。

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout

log4j.logger.com.ibatis=DEBUG

# shows SQL of prepared statements
#log4j.logger.java.sql.Connection=DEBUG

# shows parameters inserted into prepared statements
#log4j.logger.java.sql.PreparedStatement=DEBUG

# shows query results
#log4j.logger.java.sql.ResultSet=DEBUG

#log4j.logger.java.sql.Statement=DEBUG

# Console output
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

可以找到完整的Log4J文档，从Apaches 网站： Log4J 文档.

# iBATIS 调试例子：

下面的Java类是一个非常简单的例子，初始化，然后使用Java应用程序Log4J的日志库。它位于CLASSPATH中上面提到的属性文件。

```
import org.apache.log4j.Logger;

import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisUpdate{
  static Logger log = Logger.getLogger(
                      IbatisUpdate.class.getName());

  public static void main(String[] args)
   throws IOException,SQLException{
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   /* This would insert one record in Employee table. */
   log.info("Going to update record.....");
   Employee rec = new Employee();
   rec.setId(1);
   rec.setFirstName( "Roma");
   smc.update("Employee.update", rec );
   log.info("Record updated Successfully ");

   log.debug("Going to read records.....");
   List <Employee> ems = (List<Employee>)
                        smc.queryForList("Employee.getAll", null);
   Employee em = null;
   for (Employee e : ems) {
      System.out.print("  " + e.getId());
      System.out.print("  " + e.getFirstName());
      System.out.print("  " + e.getLastName());
      System.out.print("  " + e.getSalary());
      em = e;
      System.out.println("");
   }

   log.debug("Records Read Successfully ");

  }
}
```

# 编译和运行：

下面是步骤来编译并运行上述软件。请确保您已在进行的编译和执行之前，适当地设置PATH和CLASSPATH。

- 创建Employee.xml如上所示。
- 创建Employee.java如上图所示，并编译它。
- 创建IbatisUpdate.java如上图所示，并编译它。
- 创建log4j.properties文件，如上图所示。
- 执行IbatisUpdate二进制文件来运行程序。

会得到下面的结果，并记录在EMPLOYEE表进行更新, 然后相同的记录将被从EMPLOYEE表中读出。

```
DEBUG [main] - Created connection 28405330.
DEBUG [main] - Returned connection 28405330 to pool.
DEBUG [main] - Checked out connection 28405330 from pool.
DEBUG [main] - Returned connection 28405330 to pool.
  1  Roma  Ali  5000
  2  Zara  Ali  5000
  3  Zara  Ali  5000
```

# 调试方法：

在上面的例子中，我们只使用info()方法，但可以使用以下任何一种方法按你的需要：

```
public void trace(Object message);
public void debug(Object message);
public void info(Object message);
public void warn(Object message);
public void error(Object message);
public void fatal(Object message);
```

# iBATIS和Hibernate区别 - ibatis

iBatis和Hibernate之间有着较大的差异，但两者解决方案很好，因为他们有特定的领域。我个人建议使用iBATIS的，如果：

- 你想创建自己的SQL，并愿意维持他们.
- 你的环境是由关系数据模型驱动的。
- 你的项目工作有复杂架构的。

简单地要使用Hibernate，如果：

- 你的环境是由对象模型驱动的，并希望自动生成的SQL。

要计算的一些区别：

- iBATIS :
  - 简单
  - 更快的开发时间
  - 灵活
  - 封装尺寸更小
- Hibernate:
  - 为你生成SQL，这意味着你不用花时间在SQL上。
  - 提供了许多更先进的高速缓存
  - 高可扩展性

另一个区别是，iBATIS利用SQL语句可能是依赖数据库，使用Hibernate的HQL是相对独立于数据库，它是更容易改变数据库。

Hibernate映射的Java作为POJO对象，iBatis将ResultSet映射，从JDBC API给出POJO OBJETS的数据库表。

如果您使用存储过程，那么在Hibernate中可以做到这一点，但它在iBATIS比较有点困难。作为一种替代的解决方案iBATIS的映射结果集对象，所以没必要去关心表结构。这非常适用于存储过程，非常适用于报表应用程序等

最后，Hibernate和iBATIS的都是开源的对象关系映射（ORM）在同行业中可用的工具。使用这些工具的取决于你。Hibernate和iBatis两者也有来自Spring框架的良好支持，以便它不应该是一个问题，选择其中之一。

# iBATOR介绍，什么是iBATOR？ - ibatis

iBATOR是一个代码生成器，用于iBATIS。 iBATOR内部检查的一个或多个数据库表和将生成iBATIS的工件，可用于访问表。

稍后，您可以编写自定义的SQL代码或存储过程来满足您的要求。 iBATOR产生以下项目：

- SqlMap XML文件
- Java类相匹配的表的主键和字段
- 使用上述对象DAO类（可选）

iBATOR可以运行作为一个独立的JAR文件，或作为一个Ant任务，或者作为一个Eclipse插件。本教程将教你生成iBATIS的配置文件格式命令行的简单的方法。

## 下载iBATOR：

如果您使用的IDE是Eclipse，比其他下载独立的JAR。独立的JAR包括一个Ant任务来运行iBATOR，或者您也可以从Java代码中的命令行运行iBATOR。

- You can download zip file from Download iBATOR.
- You can check online documentation: iBATOR Documentation.

## Generating Configuration File:

To get up and running quickly with Abator, follow these steps:

## Step 1:

Create and fill out a configuration file ibatorConfig.xml appropriately. At a minimum, you must specify:

- A **<jdbcConnection>** element to specify how to connect to the target database.
- A **<javaModelGenerator>** element to specify target package and target project for generated Java model objects
- A **<sqlMapGenerator>** element to specify target package and target project for generated SQL map files.

- A **<daoGenerator>** element to specify target package and target project for generated DAO interfaces and classes (you may omit the <daoGenerator> element if you don't wish to generate DAOs).
- At least one database **<table>** element

**NOTE:** See the XML Configuration File Reference page for an example of an Abator configuration file.

## Step 2:

Save the file in some convenient location for example at: empibatorConfig.xml).

## Step 3:

Now run Abator from the command line with a command line as follows:

```
java -jar abator.jar -configfile     empabatorConfig.xml -overwrite
```

This will tell Abator to run using your configuration file. It will also tell Abator to overwrite any existing Java files with the same name. If you want to save any existing Java files, then omit the **-overwrite**parameter.

If there is a conflict, Abator will save the newly generated file with a unique name.

After running Abator, you will need to create or modify the standard iBATIS configuration files to make use of your newly generated code. This is explained in next section.

## Tasks After Running Abator:

After you run Abator, you will need to create or modify other iBATIS configuration artifacts. The main tasks are as follows:

- Create or Modify the SqlMapConfig.xml file.
- Create or modify the dao.xml file (only if using the iBATIS DAO Framework).

Each task is described in detail below:

## Updating the SqlMapConfig.xml File:

iBATIS uses an XML file, commonly named SqlMapConfig.xml, to specify information for a database connection, a transaction management scheme, and SQL map XML files that will be used in an iBATIS session.

Abator cannot create this file for you because Abator knows nothing about your execution environment. However, some of the items in this file relate directly to Abator generated items.

Abator specific needs in the configuration file are as follows:

- Statement namespaces must be enabled.
- Abator generated SQL Map XML files must be listed .

For example, suppose that Abator has generated an SQL Map XML file called MyTable_SqlMap.xml, and that the file has been placed in the test.xml package of your project. The SqlMapConfig.xml file should have these entries:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

  <sqlMapConfig>
    <!-- Statement namespaces are required for Abator -->
    <settings useStatementNamespaces="true" />

    <!-- Setup the transaction manager and data source that are
         appropriate for your environment
    -->
    <transactionManager type="...">
      <dataSource type="...">
      </dataSource>
    </transactionManager>

    <!-- SQL Map XML files should be listed here -->
    <sqlMap resource="test/xml/MyTable_SqlMap.xml" />

  </sqlMapConfig>
```

If there is more than one SQL Map XML file (as is quite common), then the files can be listed in any order with repeated <sqlMap> elements after the <transactionManager> element.

## Updating the dao.xml File:

The iBATIS DAO framework is configured by an xml file commonly called dao.xml.

The iBATIS DAO framework uses this file to control the database connection information for DAOs, and also to list the DAO implementation classes and DAO interfaces.

In this file you should specify the path to your SqlMapConfig.xml file, and all the Abator generated DAO interfaces and implementation classes.

For example, suppose that Abator has generated a DAO interface called MyTableDAO and a implementation class called MyTableDAOImpl, and that the files have been placed in the test.dao package of your project.

dao.xml文件应该有这些项：

```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE daoConfig
   PUBLIC "-//ibatis.apache.org//DTD DAO Configuration 2.0//EN"
   "http://ibatis.apache.org/dtd/dao-2.dtd">

  <daoConfig>
    <context>
      <transactionManager type="SQLMAP">
        <property name="SqlMapConfigResource"
                  value="test/SqlMapConfig.xml"/>
      </transactionManager>

      <!-- DAO interfaces and implementations
              should be listed here -->
      <dao interface="test.dao.MyTableDAO"
           implementation="test.dao.MyTableDAOImpl" />

    </context>
  </daoConfig>
```

**NOTE:** This step is only required if you generated DAOs for the iBATIS DAO framework.

# Jackson教程

Jackson是一个简单基于Java应用库，Jackson可以轻松的将Java对象转换成json对象和xml文档，同样也可以将json、xml转换成Java对象。Jackson所依赖的jar包较少，简单易用并且性能也要相对高些，并且Jackson社区相对比较活跃，更新速度也比较快。

## 特点

- 容易使用 - jackson API提供了一个高层次外观，以简化常用的用例。
- 无需创建映射 - API提供了默认的映射大部分对象序列化。
- 性能高 - 快速，低内存占用，适合大型对象图表或系统。
- 干净的JSON - jackson创建一个干净和紧凑的JSON结果，这是让人很容易阅读。
- 不依赖 - 库不需要任何其他的库，除了JDK。
- 开源代码 - jackson是开源的，可以免费使用。

## 三种方式处理JSON

提供了三种不同的方法来处理JSON

- 流式API - 读取并将JSON内容写入作为离散事件。 JsonParser读取数据，而JsonGenerator写入数据。它是三者中最有效的方法，是最低的开销和最快的读/写操作。它类似于Stax解析器XML。
- 树模型 - 准备JSON文件在内存里以树形式表示。 ObjectMapper构建JsonNode节点树。这是最灵活的方法。它类似于XML的DOM解析器。
- 数据绑定 - 转换JSON并从POJO（普通Java对象）使用属性访问或使用注释。它有两个类型。
    - 简单的数据绑定 - 转换JSON和Java Maps, Lists, Strings, Numbers, Booleans 和null 对象。
    - 全部数据绑定 - 转换为JSON从任何JAVA类型。

      ObjectMapper读/写JSON两种类型的数据绑定。数据绑定是最方便的方式是类似XML的JAXB解析器。

# Jackson环境安装设置 - Jackson教程

## 本地环境设置

由于Jackson是基于Java编程语言的，所以需要设置Java开发环境，这里介绍如何下载安装设置Java。请按照以下步骤来设置环境。

Java SE是免费的，点击下载链接：下载Java 根据操作系统版本下载对应版本安装。

按照说明下载java并运行.exe到机器上安装。一旦机器上安装了Java，还需要设置环境变量指向正确的安装目录：

## Setting up the path for windows 2000/XP:

假设你已经安装在 c:Program Filesjavajdk 目录：

- 在“我的电脑”右键单击并选择“属性”。
- 在“高级”选项卡下单击“环境变量”按钮。
- 现在，改变“PATH”变量，因此它也要包含Java可执行文件路径。例如，如果该路径当前设置为'C:WINDOWSSYSTEM32',那么改变路径看起来就是'C:WINDOWSSYSTEM32;c:Program Filesjavajdkin'.

## 设置为Linux，UNIX，Solaris和FreeBSD的路径：

环境变量PATH应设置为指向已安装的Java二进制文件的位置。请参考shell文件，如果这样做还搞不定的话。

例如，如果使用bash作为shell，那么将下面的行添加到最后一行 '.bashrc: export PATH=/path/to/java:$PATH'

## 一些流行的Java编辑器：

要编写Java程序，需要一个文本编辑器。在市场上可用的更复杂的IDE。但现在，可以考虑下列之一：

- Notepad: 在Windows机器上，可以使用像记事本的任何简单的文本编辑器（初学者推荐），TextPad。
- Netbeans:是一个Java IDE，它是开源和免费的，可从以下地址下载：http://www.netbeans.org/index.html.

- Eclipse: 也是一个Java IDE开发由Eclipse开源社区， 可以从 http://www.eclipse.org/ 下载【本教程推荐使用】

# 下载jackson存档

从jackson-all-1.9.0.jar.zip下载jackson jar文件的最新版本。在写这篇教程的时候，下载的是jackson-1.9.0.jar并将其复制到C:>jackson文件夹中。

| OS | 归档名称 |
| --- | --- |
| Windows | jackson-all-1.9.0.jar |
| Linux | jackson-all-1.9.0.jar |
| Mac | jackson-all-1.9.0.jar |

# 设置jackson环境

设置jackson_HOME环境变量指向Guava jar被存储在计算机上的基本目录位置。假设，我们已经提取jackson-all-1.9.0.jar，jackson文件夹不同的操作系统如下。

| OS | 输出 |
| --- | --- |
| Windows | 设置环境变量jackson_HOME到 C:jackson |
| Linux | export jackson_HOME=/usr/local/jackson |
| Mac | export jackson_HOME=/Library/jackson |

# 设置CLASSPATH变量

设置CLASSPATH环境变量指向 jackson jar 的位置。假设我们已经存储 jackson-all-1.9.0.jar 在不同的操作系统的 jackson 文件夹如下。

| OS | Output |
| --- | --- |
| Windows | 设置环境变量CLASSPATH 为%CLASSPATH%;%jackson_HOME%jackson-all-1.9.0.jar;.; |
| Linux | export CLASSPATH=$CLASSPATH:$jackson_HOME/jackson-all-1.9.0.jar:. |
| Mac | export CLASSPATH=$CLASSPATH:$jackson_HOME/jackson-all-1.9.0.jar:. |

# Jackson第一个程序 - Jackson教程

再进入学习jackson库的细节之前，让我们来看看应用程序操作功能。在这个例子中，我们创建一个Student类。将创建一个JSON字符串学生的详细信息，并将其反序列化到学生的对象，然后将其序列化到JSON字符串。

创建一个名为JacksonTester在Java类文件 C:\>Jackson_WORKSPACE.

文件*: JacksonTester.java*

```
import java.io.IOException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.map.SerializationConfig;

public class JacksonTester {
   public static void main(String args[]){
      ObjectMapper mapper = new ObjectMapper();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21}";

      //map json to student
      try {
         Student student = mapper.readValue(jsonString, Student.cla
         System.out.println(student);

         mapper.enable(SerializationConfig.Feature.INDENT_OUTPUT);
         jsonString = mapper.writeValueAsString(student);
         System.out.println(jsonString);

      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}

class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      this.age = age;
   }
   public String toString(){
      return "Student [ name: "+name+", age: "+ age+ " ]";
   }
}
```

验证结果

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Student [ name: Mahesh, age: 21 ]
{
  "name" : "Mahesh",
  "age" : 21
}
```

# 步骤 - 需要记住

以下是这里要考虑的重要步骤。

## 第1步：创建ObjectMapper对象。

创建ObjectMapper对象。它是一个可重复使用的对象。

```
ObjectMapper mapper = new ObjectMapper();
```

## 第2步：反序列化JSON到对象。

从JSON对象使用readValue()方法来获取。通过JSON字符串和对象类型作为参数JSON字符串/来源。

```
//Object to JSON Conversion
Student student = mapper.readValue(jsonString, Student.class);
```

## 第3步：序列化对象到JSON。

使用writeValueAsString()方法来获取对象的JSON字符串表示。

```
//Object to JSON Conversion
jsonString = mapper.writeValueAsString(student);
```

# Jackson ObjectMapper类 - Jackson教程

ObjectMapper类是Jackson库的主要类。它提供一些功能将转换成Java对象匹配JSON结构，反之亦然。它使用JsonParser和JsonGenerator的实例实现JSON实际的读/写。

## 类声明

以下是org.codehaus.jackson.map.ObjectMapper类的声明：

```
public class ObjectMapper
   extends ObjectCodec
      implements Versioned
```

## 嵌套类

| S.N. | 类 & 描述 |
|---|---|
| 1 | **static class ObjectMapper.DefaultTypeResolverBuilder** 定制TypeResolverBuilder，提供所谓的“默认输入”使用类型解析构建器（见enableDefaultTyping()了解详细信息）。 |
| 2 | **static class ObjectMapper.DefaultTyping** 使用enableDefaultTyping()枚举指定什么样的类型（类）默认输入应该使用。 |

## 构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **ObjectMapper()** 默认的构造函数，这将构建默认JsonFactory必要时使用StdSerializerProvider作为其SerializerProvider，并BeanSerializerFactory作为其SerializerFactory。 |
| 2 | **ObjectMapper(JsonFactory jf)** 构造使用指定的JsonFactory构建必要的JsonParsers和/或JsonGenerators映射。 |
| 3 | **ObjectMapper(JsonFactory jf, SerializerProvider sp, DeserializerProvider dp)** |
| 4 | **ObjectMapper(JsonFactory jf, SerializerProvider sp, DeserializerProvider dp, SerializationConfig sconfig, DeserializationConfig dconfig)** |
| 5 | **ObjectMapper(SerializerFactory sf)** 不推荐使用。使用其他构造来代替; 注意，可以设置序列化工厂setSerializerFactory（org.codehaus.jackson.map.SerializerFactory） |

## 继承的方法

这个类继承了以下类方法：

- java.lang.Object

# ObjectMapper示例

<a name="ad"></a>选择使用任何编辑器创建以下java程序在 C:/> Jackson_WORKSPACE

*File: JacksonTester.java*

```
import java.io.IOException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.map.SerializationConfig;

public class JacksonTester {
   public static void main(String args[]){
      ObjectMapper mapper = new ObjectMapper();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21}";

      //map json to student
      try {
         Student student = mapper.readValue(jsonString, Student.cla
         System.out.println(student);
         mapper.enable(SerializationConfig.Feature.INDENT_OUTPUT);
         jsonString = mapper.writeValueAsString(student);
         System.out.println(jsonString);

      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}

class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      this.age = age;
   }
   public String toString(){
      return "Student [ name: "+name+", age: "+ age+ " ]";
   }
}
```

验证结果

使用 javac 编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Student [ name: Mahesh, age: 21 ]
{
  "name" : "Mahesh",
  "age" : 21
}
```

# Jackson对象序列化 - Jackson教程

这里将介绍将Java对象序列化到一个JSON文件，然后再读取JSON文件获取转换为对象。在这个例子中，创建了Student类。创建将有学生对象以JSON表示在一个student.json文件。

创建一个名为JacksonTester在Java类文件在 C:\>Jackson_WORKSPACE.

文件*: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
      try {
         Student student = new Student();
         student.setAge(10);
         student.setName("Mahesh");
         tester.writeJSON(student);

         Student student1 = tester.readJSON();
         System.out.println(student1);

      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   private void writeJSON(Student student) throws JsonGenerationExc
      ObjectMapper mapper = new ObjectMapper();
      mapper.writeValue(new File("student.json"), student);
   }

   private Student readJSON() throws JsonParseException, JsonMappin
      ObjectMapper mapper = new ObjectMapper();
      Student student = mapper.readValue(new File("student.json"),
      return student;
   }
```

```
}

class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      this.age = age;
   }
   public String toString(){
      return "Student [ name: "+name+", age: "+ age+ " ]";
   }
}
```

验证结果

使用 javac 编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出结果

```
Student [ name: Mahesh, age: 10 ]
```

# Jackson数据绑定 - Jackson教程

数据绑定API用于JSON转换和使用属性访问或使用注解POJO(普通Java对象)。以下是它的两个类型。

- 简单数据绑定 - 转换JSON，从Java Maps, Lists, Strings, Numbers, Booleans 和 null 对象。
- 完整数据绑定 - 转换JSON到任何JAVA类型。我们将在下一章分别绑定。

ObjectMapper读/写JSON两种类型的数据绑定。数据绑定是最方便的方式是类似XML的JAXB解析器。

## 简单的数据绑定

简单的数据绑定是指JSON映射到Java核心数据类型。下表列出了JSON类型和Java类型之间的关系。

| Sr. No. | JSON 类型 | Java 类型 |
|---|---|---|
| 1 | object | LinkedHashMap<String,Object> |
| 2 | array | ArrayList<Object> |
| 3 | string | String |
| 4 | complete number | Integer, Long or BigInteger |
| 5 | fractional number | Double / BigDecimal |
| 6 | true \| false | Boolean |
| 7 | null | null |

让我们来看看简单的数据操作绑定。在这里，我们将映射JAVA基本类型直接JSON，反之亦然。

创建一个名为JacksonTester在Java类文件在目录 C:\>Jackson_WORKSPACE.

*File: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
```

```
import org.codehaus.jackson.map.ObjectMapper;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
         try {
            ObjectMapper mapper = new ObjectMapper();

            Map<String,Object> studentDataMap = new HashMap<String,
            int[] marks = {1,2,3};

            Student student = new Student();
            student.setAge(10);
            student.setName("Mahesh");
            // JAVA Object
            studentDataMap.put("student", student);
            // JAVA String
            studentDataMap.put("name", "Mahesh Kumar");
            // JAVA Boolean
            studentDataMap.put("verified", Boolean.FALSE);
            // Array
            studentDataMap.put("marks", marks);

            mapper.writeValue(new File("student.json"), studentData
            //result student.json
            //{
            //   "student":{"name":"Mahesh","age":10},
            //   "marks":[1,2,3],
            //   "verified":false,
            //   "name":"Mahesh Kumar"
            //}
            studentDataMap = mapper.readValue(new File("student.jso

            System.out.println(studentDataMap.get("student"));
            System.out.println(studentDataMap.get("name"));
            System.out.println(studentDataMap.get("verified"));
            System.out.println(studentDataMap.get("marks"));
      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
            e.printStackTrace();
      }
   }
}

class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
```

```
    }
    public void setName(String name) {
       this.name = name;
    }
    public int getAge() {
       return age;
    }
    public void setAge(int age) {
       this.age = age;
    }
    public String toString(){
       return "Student [ name: "+name+", age: "+ age+ " ]";
    }
}
```

验证结果

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出结果

```
{name=Mahesh, age=10}
Mahesh Kumar
false
[1, 2, 3]
```

# Jackson全数据绑定 - Jackson教程

完全数据绑定是指JSON映射到任何Java对象。

```
//Create an ObjectMapper instance
ObjectMapper mapper = new ObjectMapper();
//map JSON content to Student object
Student student = mapper.readValue(new File("student.json"), Studer
//map Student object to JSON content
mapper.writeValue(new File("student.json"), student);
```

让我们来看看简单的数据操作绑定。在这里，我们将直接映射Java对象到JSON，反之亦然。

创建一个名为JacksonTester在Java类文件目录 **C:\>Jackson_WORKSPACE**.

*File: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
      try {
         Student student = new Student();
         student.setAge(10);
         student.setName("Mahesh");
         tester.writeJSON(student);

         Student student1 = tester.readJSON();
         System.out.println(student1);

      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
```

```
   private void writeJSON(Student student) throws JsonGenerationExc
      ObjectMapper mapper = new ObjectMapper();
      mapper.writeValue(new File("student.json"), student);
   }

   private Student readJSON() throws JsonParseException, JsonMappin
      ObjectMapper mapper = new ObjectMapper();
      Student student = mapper.readValue(new File("student.json"),
      return student;
   }
}

class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      this.age = age;
   }
   public String toString(){
      return "Student [ name: "+name+", age: "+ age+ " ]";
   }
}
```

验证结果

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Student [ name: Mahesh, age: 10 ]
```

# Jackson数据绑定泛型 - Jackson教程

在简单的数据绑定中，我们使用String作为关键对象，并作为一个值对象映射类。相反，我们可以使用具体的Java对象和类型强制转换到JSON使用。

考虑下面的例子使用一个类的UserData来保存用户专用数据。

创建一个名为JacksonTester在Java类文件目录 C:\>Jackson_WORKSPACE.

文件名*: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.type.TypeReference;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
         try {
            ObjectMapper mapper = new ObjectMapper();

            Map <string, style="box-sizing: border-box;" userdata='
            UserData studentData = new UserData();
            int[] marks = {1,2,3};

            Student student = new Student();
            student.setAge(10);
            student.setName("Mahesh");
            // JAVA Object
            studentData.setStudent(student);
            // JAVA String
            studentData.setName("Mahesh Kumar");
            // JAVA Boolean
            studentData.setVerified(Boolean.FALSE);
            // Array
            studentData.setMarks(marks);
            TypeReference ref = new TypeReference<map<string,userda
            userDataMap.put("studentData1", studentData);
            mapper.writeValue(new File("student.json"), userDataMap
            //{
            //   "studentData1":
```

```
            //     {
            //        "student":
            //        {
            //            "name":"Mahesh",
            //            "age":10
            //        },
            //       "name":"Mahesh Kumar",
            //       "verified":false,
            //       "marks":[1,2,3]
            //   }
            //}
            userDataMap = mapper.readValue(new File("student.json")

            System.out.println(userDataMap.get("studentData1").getS
            System.out.println(userDataMap.get("studentData1").getN
            System.out.println(userDataMap.get("studentData1").getV
            System.out.println(Arrays.toString(userDataMap.get("stu
      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
            e.printStackTrace();
      }
   }
}

class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      this.age = age;
   }
   public String toString(){
      return "Student [ name: "+name+", age: "+ age+ " ]";
   }
}

class UserData {
   private Student student;
   private String name;
   private Boolean verified;
   private int[] marks;
```

```
   public UserData(){}

   public Student getStudent() {
      return student;
   }
   public void setStudent(Student student) {
      this.student = student;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public Boolean getVerified() {
      return verified;
   }
   public void setVerified(Boolean verified) {
      this.verified = verified;
   }
   public int[] getMarks() {
      return marks;
   }
   public void setMarks(int[] marks) {
      this.marks = marks;
   }
}</map<string,userdata></string,></string,>
```

验证输出

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Student [ name: Mahesh, age: 10 ]
Mahesh Kumar
false
[1, 2, 3]
```

# Jackson树模型 - Jackson教程

树模型准备JSON文件的内存树表示。 ObjectMapper构建JsonNode节点树。这是最灵活的方法。它类似于DOM解析器的XML。

## 从JSON创建树

ObjectMapper提供一个指针树的根节点在读取JSON之后。根节点可用于遍历完全树。考虑下面的代码片段获得提供JSON字符串的根节点。

```
//Create an ObjectMapper instance
ObjectMapper mapper = new ObjectMapper();
String jsonString = "{\"name\":\"Mahesh Kumar\", \"age\":21,\"verif
//create tree from JSON
JsonNode rootNode = mapper.readTree(jsonString);
```

## 遍历树模型

使用相对路径来根节点在遍历树，并处理该数据得到的每个节点。考虑下面的代码片段遍历提供的根节点的树。

```
JsonNode nameNode = rootNode.path("name");
System.out.println("Name: "+ nameNode.getTextValue());

JsonNode marksNode = rootNode.path("marks");
Iterator <jsonnode style="box-sizing: border-box;">iterator = marks
```

## 示例

创建一个名为JacksonTester在Java类文件目录 C:\>Jackson_WORKSPACE.

*File: JacksonTester.java*

```
import java.io.IOException;
import java.util.Iterator;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
      try {
         ObjectMapper mapper = new ObjectMapper();
         String jsonString = "{\"name\":\"Mahesh Kumar\", \"age\":2
         JsonNode rootNode = mapper.readTree(jsonString);

         JsonNode nameNode = rootNode.path("name");
         System.out.println("Name: "+ nameNode.getTextValue());

         JsonNode ageNode = rootNode.path("age");
         System.out.println("Age: " + ageNode.getIntValue());

         JsonNode verifiedNode = rootNode.path("verified");
         System.out.println("Verified: " + (verifiedNode.getBoolean

         JsonNode marksNode = rootNode.path("marks");
         Iterator<JsonNode> iterator = marksNode.getElements();
         System.out.print("Marks: [ ");
         while (iterator.hasNext()) {
            JsonNode marks = iterator.next();
            System.out.print(marks.getIntValue() + " ");
         }
         System.out.println("]");
      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

验证输出结果

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Name: Mahesh Kumar
Age: 21
Verified: No
Marks: [ 100 90 85 ]
```

# 树到JSON转换

在这个例子中，我们已经使用JsonNode并将其写入到一个JSON文件，并读回创建了一棵树。

创建一个名为JacksonTester在Java类文件目录 C:\>Jackson_WORKSPACE.

*File: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;
import java.util.Iterator;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.node.ArrayNode;
import org.codehaus.jackson.node.ObjectNode;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
      try {
         ObjectMapper mapper = new ObjectMapper();

         JsonNode rootNode = mapper.createObjectNode();
         JsonNode marksNode = mapper.createArrayNode();
         ((ArrayNode)marksNode).add(100);
         ((ArrayNode)marksNode).add(90);
         ((ArrayNode)marksNode).add(85);
         ((ObjectNode) rootNode).put("name", "Mahesh Kumar");
         ((ObjectNode) rootNode).put("age", 21);
         ((ObjectNode) rootNode).put("verified", false);
         ((ObjectNode) rootNode).put("marks",marksNode);

         mapper.writeValue(new File("student.json"), rootNode);
```

```
         rootNode = mapper.readTree(new File("student.json"));

         JsonNode nameNode = rootNode.path("name");
         System.out.println("Name: "+ nameNode.getTextValue());

         JsonNode ageNode = rootNode.path("age");
         System.out.println("Age: " + ageNode.getIntValue());

         JsonNode verifiedNode = rootNode.path("verified");
         System.out.println("Verified: " + (verifiedNode.getBoolean

         JsonNode marksNode1 = rootNode.path("marks");
         Iterator<JsonNode> iterator = marksNode1.getElements();
         System.out.print("Marks: [ ");
         while (iterator.hasNext()) {
            JsonNode marks = iterator.next();
            System.out.print(marks.getIntValue() + " ");
         }
         System.out.println("]");
      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

验证结果

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Name: Mahesh Kumar
Age: 21
Verified: No
Marks: [ 100 90 85 ]
```

# 从树到Java对象转换

在这个例子中，我们已经使用JsonNode并将其写入到一个JSON文件，并回读然后将一个Student对象其转换为创建了一棵树。

创建一个名为JacksonTester在Java类文件目录 C:\>Jackson_WORKSPACE.

*File: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;
import java.util.Arrays;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.node.ArrayNode;
import org.codehaus.jackson.node.ObjectNode;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
      try {
         ObjectMapper mapper = new ObjectMapper();

         JsonNode rootNode = mapper.createObjectNode();
         JsonNode marksNode = mapper.createArrayNode();
         ((ArrayNode)marksNode).add(100);
         ((ArrayNode)marksNode).add(90);
         ((ArrayNode)marksNode).add(85);
         ((ObjectNode) rootNode).put("name", "Mahesh Kumar");
         ((ObjectNode) rootNode).put("age", 21);
         ((ObjectNode) rootNode).put("verified", false);
         ((ObjectNode) rootNode).put("marks",marksNode);

         mapper.writeValue(new File("student.json"), rootNode);

         rootNode = mapper.readTree(new File("student.json"));

         Student student = mapper.treeToValue(rootNode, Student.cla

         System.out.println("Name: "+ student.getName());
         System.out.println("Age: " + student.getAge());
         System.out.println("Verified: " + (student.isVerified() ?
         System.out.println("Marks: "+Arrays.toString(student.getMa
      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
```

```
        }
    }
}

class Student {
    String name;
    int age;
    boolean verified;
    int[] marks;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public boolean isVerified() {
        return verified;
    }
    public void setVerified(boolean verified) {
        this.verified = verified;
    }
    public int[] getMarks() {
        return marks;
    }
    public void setMarks(int[] marks) {
        this.marks = marks;
    }
}
```

验证结果

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Name: Mahesh Kumar
Age: 21
Verified: No
Marks: [ 100 90 85 ]
```

# Jackson流式API - Jackson教程

流式API读取和写入JSON内容离散事件。 JsonParser读取数据，而JsonGenerator写入数据。它是三者中最有效的方法，是最低开销和最快的读/写操作。它类似于XML的Stax解析器。

在本文中，我们将展示的使用Jackson的流式API 读写JSON数据。流式API工作使用JSON为每一个细节的都是要小心处理。下面的例子将使用两个类：

- JsonGenerator - 写入JSON字符串。
- JsonParser - 解析JSON字符串。

## 使用JsonGenerator写入JSON

使用JsonGenerator是非常简单的。首先使用JsonFactory.createJsonGenerator()方法创建一个JsonGenerator，并用write*()方法来写每一个JSON值。

```
JsonFactory jasonFactory = new JsonFactory();
JsonGenerator jsonGenerator = jasonFactory.createJsonGenerator(new
   "student.json"), JsonEncoding.UTF8);
// {
jsonGenerator.writeStartObject();
// "name" : "Mahesh Kumar"
jsonGenerator.writeStringField("name", "Mahesh Kumar");
```

让我们来看看JsonGenerator操作。创建一个名为JacksonTester的Java类文件在目录 C:\>Jackson_WORKSPACE.

*File: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;
import java.util.Map;

import org.codehaus.jackson.JsonEncoding;
import org.codehaus.jackson.JsonFactory;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
```

```
      try {
         JsonFactory jasonFactory = new JsonFactory();

         JsonGenerator jsonGenerator = jasonFactory.createJsonGener
            "student.json"), JsonEncoding.UTF8);
         // {
         jsonGenerator.writeStartObject();
         // "name" : "Mahesh Kumar"
         jsonGenerator.writeStringField("name", "Mahesh Kumar");
         // "age" : 21
         jsonGenerator.writeNumberField("age", 21);
         // "verified" : false
         jsonGenerator.writeBooleanField("verified", false);
         // "marks" : [100, 90, 85]
         jsonGenerator.writeFieldName("marks");
         // [
         jsonGenerator.writeStartArray();
         // 100, 90, 85
         jsonGenerator.writeNumber(100);
         jsonGenerator.writeNumber(90);
         jsonGenerator.writeNumber(85);
         // ]
         jsonGenerator.writeEndArray();
         // }
         jsonGenerator.writeEndObject();
         jsonGenerator.close();

         //result student.json
         //{
         //   "name":"Mahesh Kumar",
         //   "age":21,
         //   "verified":false,
         //   "marks":[100,90,85]
         //}
         ObjectMapper mapper = new ObjectMapper();
         Map<String,Object> dataMap = mapper.readValue(new File("st

         System.out.println(dataMap.get("name"));
         System.out.println(dataMap.get("age"));
         System.out.println(dataMap.get("verified"));
         System.out.println(dataMap.get("marks"));
      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

验证结果

使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行jacksonTester看到的结果：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Mahesh Kumar
21
false
[100, 90, 85]
```

# 使用JsonParser 读取JSON

使用JsonParser是非常简单的。首先使用JsonFactory.createJsonParser()方法创建JsonParser，并使用它的nextToken()方法来读取每个JSON字符串作为标记。检查每个令牌和相应的过程。

```
JsonFactory jasonFactory = new JsonFactory();
JJsonParser jsonParser = jasonFactory.createJsonParser(new File("st
while (jsonParser.nextToken() != JsonToken.END_OBJECT) {
   //get the current token
   String fieldname = jsonParser.getCurrentName();
   if ("name".equals(fieldname)) {
      //move to next token
      jsonParser.nextToken();
      System.out.println(jsonParser.getText());
   }
}
```

让我们来看看JsonParser的操作。创建一个名为JacksonTester在Java类在文件夹C:\>Jackson_WORKSPACE.

文件*: JacksonTester.java*

```
import java.io.File;
import java.io.IOException;

import org.codehaus.jackson.JsonEncoding;
```

```
import org.codehaus.jackson.JsonFactory;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.JsonMappingException;

public class JacksonTester {
   public static void main(String args[]){
      JacksonTester tester = new JacksonTester();
      try {
         JsonFactory jasonFactory = new JsonFactory();

         JsonGenerator jsonGenerator = jasonFactory.createJsonGene
            "student.json"), JsonEncoding.UTF8);
         jsonGenerator.writeStartObject();
         jsonGenerator.writeStringField("name", "Mahesh Kumar");
         jsonGenerator.writeNumberField("age", 21);
         jsonGenerator.writeBooleanField("verified", false);
         jsonGenerator.writeFieldName("marks");
         jsonGenerator.writeStartArray(); // [
         jsonGenerator.writeNumber(100);
         jsonGenerator.writeNumber(90);
         jsonGenerator.writeNumber(85);
         jsonGenerator.writeEndArray();
         jsonGenerator.writeEndObject();
         jsonGenerator.close();

         //result student.json
         //{
         //   "name":"Mahesh Kumar",
         //   "age":21,
         //   "verified":false,
         //   "marks":[100,90,85]
         //}

         JsonParser jsonParser = jasonFactory.createJsonParser(new
         while (jsonParser.nextToken() != JsonToken.END_OBJECT) {
            //get the current token
            String fieldname = jsonParser.getCurrentName();
            if ("name".equals(fieldname)) {
               //move to next token
               jsonParser.nextToken();
               System.out.println(jsonParser.getText());
            }
            if("age".equals(fieldname)){
               //move to next token
               jsonParser.nextToken();
               System.out.println(jsonParser.getNumberValue());
            }
            if("verified".equals(fieldname)){
               //move to next token
               jsonParser.nextToken();
```

```
                System.out.println(jsonParser.getBooleanValue());
            }
            if("marks".equals(fieldname)){
                //move to [
                jsonParser.nextToken();
                // loop till token equal to "]"
                while (jsonParser.nextToken() != JsonToken.END_ARRAY
                    System.out.println(jsonParser.getNumberValue());
                }
            }
         }
      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

验证结果

编译使用javac编译如下类：

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

现在运行 jacksonTester 看到结果如下：

```
C:\Jackson_WORKSPACE>java JacksonTester
```

验证输出

```
Mahesh Kumar
21
false
[100, 90, 85]
```

# JasperReports教程




Jasper报表是一个开源的Java报表引擎，是基于Java的，它没有自己的表达式语法。

由于JasperReports是一个Java类库，而不是针对最终用户，而是有针对那些需要报表功能添加到自己的应用程序的Java开发人员。

## 读者

本教程为Java开发那些愿意充实自己的报表技术而编制。完成本教程后，对JasperReports熟悉程序达到中等水平。

## 前提条件

在继续本教程，应该对Java编程语言的一个很好的理解。还需要对Apache Ant的基本理解，因为所有的例子都被编译和执行中使用ANT。

### 1 - Java教程

如果想学习JAVA入门，推荐以下教程

Java教程

### 2 - ANT教程

如果想使用Apache Ant，那么可以通过以教程学习。

ANT教程

# JasperReports入门，JasperReports是什么？ - JasperReports教程

## 什么是报表

报表是从数据库中的数据有意义和良好汇总信息。通常情况下，日常活动是自动化和数据汇总到一个决策支持的格式“报告”。报表作为奇迹，当一般的杂乱数据转换成迷人的图表，图形和其他图形表示转换。

## 报表模板

一般有以下报表布局之后是许多商业报告生成工具来生成报告。

TITLE
PAGEHEADER
COLUMNHEADER
DETAIL
COLUMNFOOTER
PAGEFOOTER
SUMMARY


以下是图中提到的每个元件的描述。

| 元素 | 描述 |
| --- | --- |
| title | 标题包含该报告的标题。它只会出现一次，在报告的一开始，例如，“Yiibai教程报告”。 |
| pageHeader | PageHeader可能包含日期和时间的信息和/或组织的名称。这出现在每个页面的顶部。 |
| columnHeader | ColumnHeader列出了要在报表中显示，例如，“作者姓名”，“启动时间”，“完成时间”具体字段，“工作时间”和“日期”等的名称。 |
| detail | 详细信息在这里显示的特定字段（在的columnHeader列表）条目，例如部分"Manisha", "9:00", "18:00", "9", "10.02.2013". |
| columnFooter | ColumnFooter可以显示的任何字段的总和，例如，"Total Hours Worked: 180" |
| pageFooter | PageFooter可能包含页面计数信息。它出现在每个页面的底部，例如，"1/23". |
| summary | 摘要包含从“细节”部分推断出的信息，例如，工作小时为每个作者的数量列表，总工时为每个作者可以把视力表像饼图，曲线图等，为更好的比较。 |

# Jasper报表

报表开发过程中面临的常见故障归纳在以下几点：

- 核心变化：为了反映业务发生变化或改进它通常以改变报告的核心逻辑。
- 结果输出：有各种各样的格式，报表可导出到如：HTML，文本，PDF，MS Excel，RTF，ODT，逗号分隔值，XML或图像。
- 复杂报表：子报表和交叉表报告是很好的例子。
- 图表报表：视觉图为例。图，饼图，XY折线图，条形图，仪表和时间序列

为了消除上述点的开销，并促进报告过程中，很多框架，工具，库和第三方应用进行了介绍。Jasper Report 报表是其中之一。

Jasper Report 是一个开源的Java报表引擎，它不像其他的报表工具，例如Crystal报表是基于Java的，没有自己的表达式语法。JasperReports有提供丰富的内容到屏幕上，到打印机，或转换成PDF，HTML，XLS，RTF，ODT，CSV，TXT和XML文件的能力。因为它不是一个独立的工具，它不能被安装在其自身。相反，它是由包括它在应用程序的CLASSPATH库嵌入到Java应用程序。

JasperReports是一个Java类库，而不是针对最终用户，而是有针对性的对谁需要的报表功能添加到自己的应用程序的Java开发人员。

# JasperReports的特点

一些主要的 JasperReport 的功能包括：

- 具有灵活的报表布局。
- 它可以用文字或图形显示数据。
- 开发人员可以通过多种方式提供数据。
- 它可以接受来自多个数据源的数据。
- 它可以生成水印（水印是这样的方式被放置在主图像的副图像）
- 它可以生成子报表。
- 它能够导出报表到多种格式的。

# JasperReport环境设置 - JasperReports教程

JasperReport是一个纯Java库，而不是一个独立的应用程序。它不能单独运行，因此它需要被嵌入到另一个客户端或服务器端的Java应用程序。因为它是基于Java，它可以在任何支持Java的平台（JDK1.3及以上）上运行。所有JasperReports的功能是聚集在一个JAR文件中，一般名称为：jasperreports-x.x.x.jar。JasperReport库链接：这个JAR文件必需及可选库（压缩文件）可以从网站上下载。下载最新的版本。

ZIP文件包含JasperReports源代码，相关JAR和大量的实例演示JasperReport功能以及JasperReportsJAR文件。

## JasperReport环境配置

要开始创建我们需要设置环境准备。解压缩下载的JasperReport ZIP文件到任何位置（在我们的例子中，我们已经提取到C: oolsjasperreports-5.0.1）。解压缩文件的目录结构如下图示：

Address C:\tools\jasperreports-5.0.1

| Name | Size | Type | Date Modified |
|---|---|---|---|
| build | | File Folder | 1/3/2013 2:33 PM |
| demo | | File Folder | 1/3/2013 2:35 PM |
| dist | | File Folder | 1/3/2013 2:33 PM |
| docs | | File Folder | 1/3/2013 2:35 PM |
| lib | | File Folder | 1/3/2013 2:35 PM |
| src | | File Folder | 1/3/2013 2:35 PM |
| .classpath | 3 KB | CLASSPATH File | 11/23/2012 4:41 PM |
| .project | 1 KB | PROJECT File | 11/23/2012 4:41 PM |
| build.xml | 25 KB | XML Document | 11/23/2012 4:48 PM |
| changes.txt | 58 KB | Text Document | 1/3/2013 2:01 PM |
| license.txt | 8 KB | Text Document | 11/23/2012 4:41 PM |
| pom.xml | 13 KB | XML Document | 1/3/2013 1:35 PM |
| readme.txt | 3 KB | Text Document | 11/23/[illegible] |



下面是所有目录的详细信息：

- build: 包含已编译的JasperReport类文件。
- demo: 包含演示JasperReports功能几个方面的各种例子。
- dist: 包含的JasperReports-x.x.x.jar文件。将这个JAR文件添加到CLASSPATH。
- docs: 包含了JasperReports的文档的本地副本。
- lib: 包含所需的所有JAR文件，这样既可以建立JasperReports，并把它用在我们的应用程序。
- src: 包含了JasperReports的源代码。

- build.xml: Ant构建文件构建JasperReports的源代码。如果不打算修改JasperReports，并不需要使用这个文件，因为JasperReports发布成已编译的形式。
- changes.txt: 一个文本文件解释的JasperReports类库的当前和以前版本之间的差异。
- license.txt: 包含LGPL（较宽松通用公共许可证）许可的全文文本文档。
- readme.txt: 一个文本文件包含有关如何建立和执行提供的示例说明。

基本上，我们只使用jasperreports-x.x.x.jar 在lib目录下的路程和JAR下生成报表。由于Jasper报表作为一个开源的工具，如果任何缺陷或错误执行的jasperreports-x.x.x.jar中是公认的，我们可以修复它并使用build.xml文件再次生成的JAR文件。

## 设置CLASSPATH

要使用JasperReport，需要设置下列文件到CLASSPATH中：

- jasperreports-x.x.x.jar, 其中x.x.x是JasperReports的版本。此目录下找到 C: oolsjasperreports-x.x.xdist).
- lib子目录中的所有JAR文件 (C: oolsjasperreports-x.x.xlib).

在安装的时候，我们使用JasperReport5.0.1版本。在“我的电脑”右键单击并选择“属性”，“高级”选项卡下单击“环境变量”按钮。现在有了这个更新的“路径”变量添加：C: oolsjasperreports-5.0.1distjasperreports-5.0.1.jar;C: oolsjasperreports-5.0.1lib;. 现在，可以创建报表了。

> 在本教程中的所有例子中，已经使用Ant任务来生成报告。构建文件会自己负责，包括所有所需的JAR生成报告。因此，设定上述的CLASSPATH只会帮助生成报表，而无需使用ANT。

## 生成安装

在本教程中的所有例子：

- 使用简单的文本编辑器写入。
- 已保存的目录下 C: oolsjasperreports-5.0.1 estsrccomyiibai.
- 已编译并从命令提示符下执行，使用Apache Ant。我们将使用它我们将在Ant build.xml文件中的后续章节中导入abaseBuild.xml文件。将此文件保存到C: oolsjasperreports-5.0.1 est. 以下是baseBuild.xml文件的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportExample" basedir=".">
   <description>Previews our JasperReport XML Design</description>
   <property name="file.name" value="jasper_report_template" />
   <!-- Directory where the JasperReports project file was extracte
   needs to be changed to match the local environment -->
   <property name="jasper.dir" value="../" />
   <property name="dist.dir" value="${jasper.dir}/dist" />
   <property name="lib.dir" value="${jasper.dir}/lib" />
   <property name="src.dir" value="src" />
   <property name="classes.dir" value="classes" />
   <property name="main-class" value="com.yiibai.HelpMe" />

   <path id="classpath">
      <pathelement location="./" />
      <pathelement location="${classes.dir}" />
      <fileset dir="${lib.dir}">
         <include name="**/*.jar" />
      </fileset>
      <fileset dir="${dist.dir}">
         <include name="**/*.jar" />
      </fileset>
   </path>

   <target name="compile" depends="clean-sample">
   <mkdir dir="${classes.dir}"/>
   <javac srcdir="${src.dir}" destdir="${classes.dir}"
      classpathref="classpath" />
   </target>

   <target name="run" depends="compile">
      <echo message="Running class : ${main-class}"/>
      <java fork="true" classname="${main-class}">
         <classpath>
            <path refid="classpath" />
         </classpath>
      </java>
   </target>
   <target name="clean-sample">
      <delete dir="${classes.dir}" />
      <delete file="./${file.name}.jasper" />
      <delete file="./${file.name}.jrprint" />
   </target>
</project>
```

此文件具有所有必需的目标，比如清除目录，编译java文件，并执行类文件。

以下是baseBuild.xml提到的细节不同的目录。假设当前目录是 C:
oolsjasperreports-5.0.1 est):

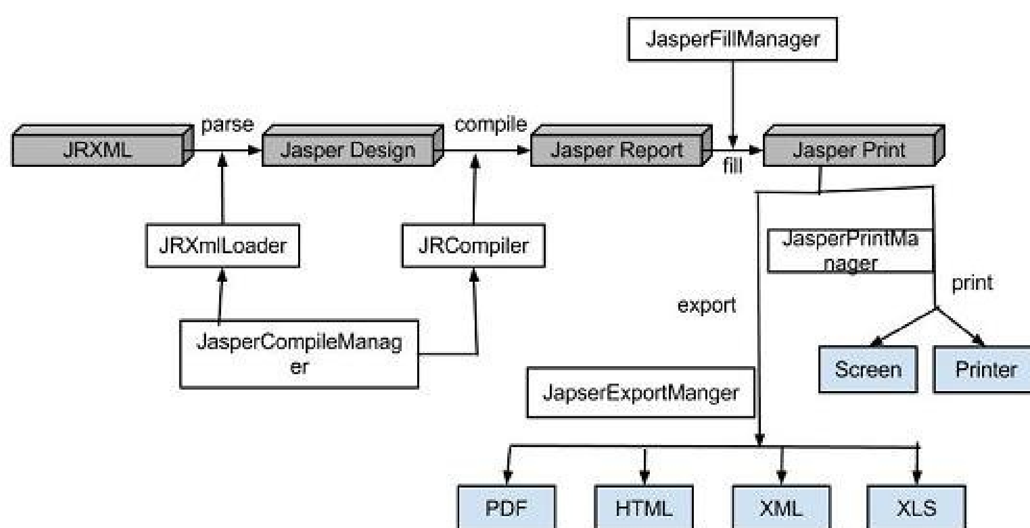
- jasper.dir: 是 C: oolsjasperreports-5.0.1 directory
- lib.dir: 是 C: oolsjasperreports-5.0.1lib directory
- src.dir: 是 C: oolsjasperreports-5.0.1 estsrc
- classes.dir: 是 C: oolsjasperreports-5.0.1 estclasses
- main-class: com.yiibai.HelpMe. 这个类执行，如果没有类文件名是在命令行中通过一个简单的类。将此文件保存到 C: oolsjasperreports-5.0.1 estsrccomyiibai.

```
package com.yiibai;

public class HelpMe {
    public static void main(String[] args) {
        System.out.println("This is the default class executed."
            + "Please pass the fully qualified class"
            + " name to be executed as command line"
            + " parameter, for example,"
            + " com.yiibai.HelpMe ");
    }
}
```

# Jasper 管理类

还有的类，它们将被用来编译JRXML报表设计，以填补报表，打印报表，导出为PDF，HTML和XML文件，查看生成的报表和报表设计序号。




这些类的列表是：

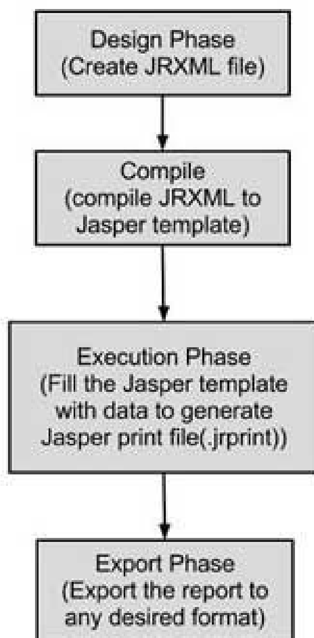
- *net.sf.jasperreports.engine.JasperCompileManager:* 用于编译JRXML报表模板。
- *net.sf.jasperreports.engine.JasperFillManager:* 用于填充一个报表，从数据源的数据
- *net.sf.jasperreports.engine.JasperPrintManager:* 用于打印的JasperReports类库生成的文件
- *net.sf.jasperreports.engine.JasperExportManager:* 用于获取PDF，HTML或XML内容以供报表填充过程中产生的文件
- *net.sf.jasperreports.view.JasperViewer:* 它代表了一个简单的Java Swing应用程序，可以加载和显示报表。
- *net.sf.jasperreports.view.JasperDesignViewer:* 用于在设计时预览报表模板。

## 设置Apache ANT

我们将构建所有的例子使用Apache Ant。所以请检查ANT - 设置Apache Ant环境。

# JasperReport生命周期 - JasperReports教程

JasperReports的主要目的是为了在一个简单而灵活的方式创建页面为导向，准备好打印文档。下面的流程图描述了一个典型的工作流程，同时创建报表。




如在图片的生命周期具有以下明显的阶段

1. 设计报表在这一步中，我们创建JRXML文件，该文件是包含的报表布局定义的XML文档。我们可以使用任何文本编辑器或iReportDesigner手动创建它。如果iReportDesigner使用的布局被设计成可视化的方式时，JRXML实际的结构可以被忽略。

2. 编译报表 在这一步中JRXML被编译为二进制对象称为Jasper文件(*.jasper)。做此编译是出于性能方面的考虑。Jasper文件是什么？它需要随应用程序以运行报表。

3. 执行报表（数据填充到报表） 在该步骤中从应用程序数据被填充在已编译的报表。类net.sf.jasperreports.engine.JasperFillManager提供了必要的功能，填补了报告中的数据。Jasper打印文件 (*.jrprint) 被创建，它可以用来打印或者导出报告。

4. 导出报表到所需的格式 在这一步中，我们可以导出在上一步中使用JasperExportManager任何格式创建的Jasper打印文件。由于Jasper 提供各种形式的导出，因此具有相同的输入，我们可以创建数据的多种表示形式。

上述每个步骤的详细介绍将在以后的章节中解释。

# JasperReport报表设计 - JasperReports教程

在JRXML模板（或JRXML文件）中的JasperReport 都是标准的 XML文件，以.JRXML扩展。所有JRXML文件包含标签<jasperReport>，作为根元素。这反过来又包含许多子元素（所有这些都是可选的）。JasperReport框架，可以处理不同类型的数据源。在本教程中，我们将展示如何生成一个基本的报表，只是通过传递Java数据对象（使用Java Bean）集合传给JasperReport引擎。最后报表应显示的人的名单的名字和国家。

本章介绍如何设计一个JasperReport。下面的步骤将在本章中：

- 创建一个JRXML报表模板。
- 预览XML报表模板。

## 创建一个JRXML报表模板

创建JRXML文件，该文件是jasper_report_template.jrxml使用文本编辑器，并保存此文件按照我们的环境设置，在 C: oolsjasperreports-5.0.1 est 。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">
    <queryString>
    <![CDATA[]]>
    </queryString>
    <field name="country" class="java.lang.String">
        <fieldDescription><![CDATA[country]]></fieldDescription>
    </field>
    <field name="name" class="java.lang.String">
        <fieldDescription><![CDATA[name]]></fieldDescription>
    </field>
    <columnHeader>
     <band height="23">
    <staticText>
        <reportElement mode="Opaque" x="0" y="3" width="535"
         height="15" backcolor="#70A9A9" />
        <box>
            <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
        </box>
        <textElement />
```

```
        <text><![CDATA[]]> </text>
    </staticText>
    <staticText>
        <reportElement x="414" y="3" width="121" height="15" />
        <textElement textAlignment="Center"
            verticalAlignment="Middle">
        <font isBold="true" />
        </textElement>
        <text><![CDATA[Country]]></text>
    </staticText>
    <staticText>
        <reportElement x="0" y="3" width="136" height="15" />
        <textElement textAlignment="Center"
            verticalAlignment="Middle">
        <font isBold="true" />
        </textElement>
        <text><![CDATA[Name]]></text>
    </staticText>
     </band>
    </columnHeader>
    <detail>
        <band height="16">
     <staticText>
         <reportElement mode="Opaque" x="0" y="0" width="535"
         height="14" backcolor="#E5ECF9" />
         <box>
                <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
         </box>
         <textElement />
         <text><![CDATA[]]> </text>
     </staticText>
     <textField>
         <reportElement x="414" y="0" width="121" height="15" />
         <textElement textAlignment="Center"
             verticalAlignment="Middle">
         <font size="9" />
         </textElement>
         <textFieldExpression class="java.lang.String">
             <![CDATA[$F{country}]]>
        </textFieldExpression>
     </textField>
     <textField>
         <reportElement x="0" y="0" width="136" height="15" />
         <textElement textAlignment="Center"
             verticalAlignment="Middle" />
             <textFieldExpression class="java.lang.String">
            <![CDATA[$F{name}]]>
             </textFieldExpression>
     </textField>
        </band>
    </detail>
</jasperReport>
```

下面是在上述报表模板主要字段的详细信息：

- <queryString>: 这是空的（因为我们传递的数据通过Java Beans）。通常包含以检索报表结果的SQL语句。
- <field name>: 此元素用于从数据源或查询数据映射到报表模板。name是重复使用到报表主体（它们大小写敏感）。
- <fieldDescription>: 此元素的映射字段名称与XML文件中的相应元素。
- <staticText>: 这个定义不依赖于任何数据源，变量，参数或报表表达式静态文本。
- <textFieldExpression>: 这定义结果字段的外观。
- $F{country}: 这是一个包含结果的预定义字段的标签<field name>的变量的值。
- <band>: 包含显示在报表中的数据。

一旦报表设计已准备就绪，将其保存在C: 目录。

## 预览XML报表模板

有提供的实用工具net.sf.jasperreports.view.JasperDesignViewer在JasperReports的JAR文件，这有助于预览报表设计，而无需编译或填充它。此实用程序是一个独立的Java应用程序，因此可以使用ANT执行。

让我们来写一个Ant目标viewDesignXML查看JRXML。因此，让我们在C:
oolsjasperreports-5.0.1 est目录创建和保存build.xml（应放置在JRXML在同一个目录下）。这里是build.xml文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewDesignXML" basedir=".

    <import file="baseBuild.xml" />
    <target name="viewDesignXML" description="Design viewer is laun
    to preview the JXML report design.">
     <java classname= "net.sf.jasperreports.view.JasperDesignViewer
        fork="true">
        <arg value="-XML" />
        <arg value="-F${file.name}.jrxml" />
        <classpath refid="classpath" />
     </java>
    </target>

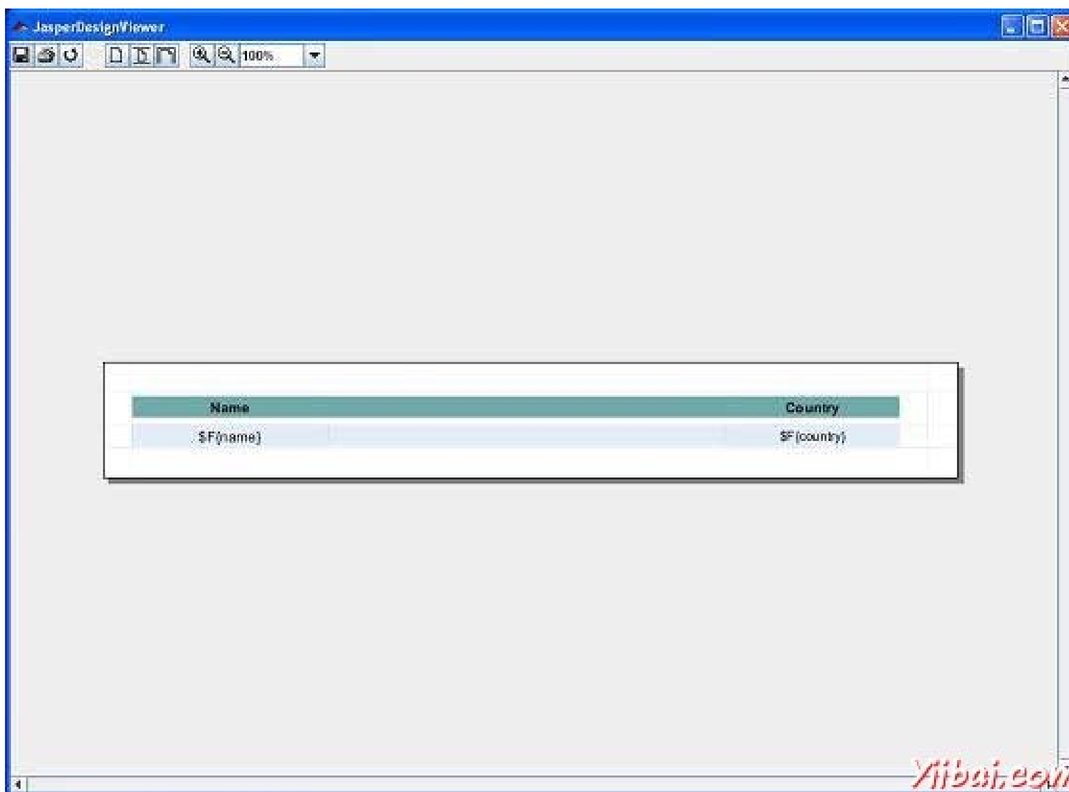
</project>
```

接下来，让我们打开命令提示符并转到build.xml文件放置的目录。执行命令ant（由于viewDesignXML是默认的目标）。输出如下：

```
C:    oolsjasperreports-5.0.1    est>ant
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

viewDesignXML:
[java] log4j:WARN No appenders could be found for logger
(net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
[java] log4j:WARN Please initialize the log4j system properly.
```

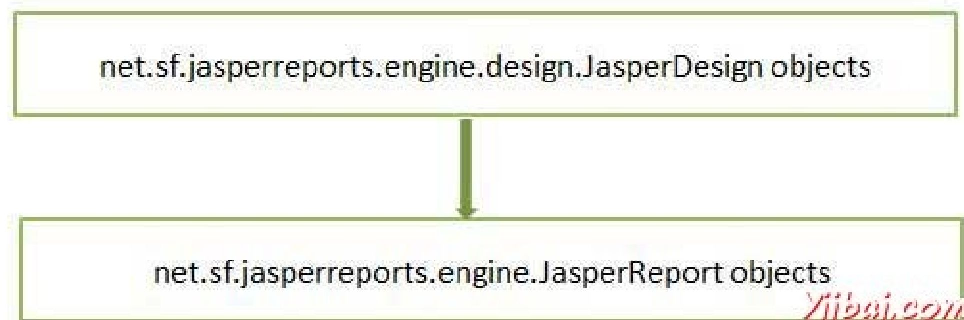
Log4j的警告可以忽略，如上面执行的结果，一个标有“JasperDesignViewer”窗口打开，显示我们的报表模板的预览。




正如我们看到获得的数据只报表表达式显示，作为JasperDesignViewer没有获得实际的数据源或报表参数。通过关闭窗口或按下Ctrl-C在命令行窗口终止JasperDesignViewer。

# JasperReport编译报表设计 - JasperReports教程

我们在前面的章节中产生的JasperReport模板（JRXML文件）。这个文件不能直接用于生成报告。它必须被编译成JasperReport的“本地二进制"格式，称为Jasperfile。在编制我们把JasperDesign对象转换成JasperReport的对象：




接口net.sf.jasperreports.engine.design.JRCompiler编译过程中起着核心的一部分。这个接口有根据用于报表表达式语言，它可以只要编译器可以实现在运行时计算它被用Java编写的，Groovy，JavaScript的或任何其他脚本语言的几个实现。我们可以通过以下两种方式编译JRXML文件：

1. 提供编程编译。
2. 编译通过ANT任务。

## JRXML提供编程编译

JasperReports的API提供了一个门面类net.sf.jasperreports.engine.JasperCompileManager用于编译JasperReport。这个类包含几个公共静态方法编制的报告模板。模板的源可以从文件，输入流，内存中的对象。

该jrxml文件（jasper_report_template.jrxml）的内容如下。它被保存在目录 **C: oolsjasperreports-5.0.1 est:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">

    <queryString>
```

```
     <![CDATA[]]>
     </queryString>
     <field name="country" class="java.lang.String">
         <fieldDescription><![CDATA[country]]></fieldDescription>
     </field>
     <field name="name" class="java.lang.String">
         <fieldDescription><![CDATA[name]]></fieldDescription>
     </field>
     <columnHeader>
      <band height="23">
     <staticText>
         <reportElement mode="Opaque" x="0" y="3" width="535"
          height="15" backcolor="#70A9A9" />
         <box>
             <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
         </box>
         <textElement />
         <text><![CDATA[]]> </text>
     </staticText>
     <staticText>
         <reportElement x="414" y="3" width="121" height="15" />
         <textElement textAlignment="Center"
             verticalAlignment="Middle">
         <font isBold="true" />
         </textElement>
         <text><![CDATA[Country]]></text>
     </staticText>
     <staticText>
         <reportElement x="0" y="3" width="136" height="15" />
         <textElement textAlignment="Center"
             verticalAlignment="Middle">
         <font isBold="true" />
         </textElement>
         <text><![CDATA[Name]]></text>
     </staticText>
      </band>
     </columnHeader>
     <detail>
         <band height="16">
      <staticText>
          <reportElement mode="Opaque" x="0" y="0" width="535"
          height="14" backcolor="#E5ECF9" />
          <box>
                 <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
          </box>
          <textElement />
          <text><![CDATA[]]> </text>
      </staticText>
      <textField>
          <reportElement x="414" y="0" width="121" height="15" />
          <textElement textAlignment="Center"
              verticalAlignment="Middle">
          <font size="9" />
```

```
            </textElement>
            <textFieldExpression class="java.lang.String">
                <![CDATA[$F{country}]]>
          </textFieldExpression>
       </textField>
       <textField>
            <reportElement x="0" y="0" width="136" height="15" />
            <textElement textAlignment="Center"
                verticalAlignment="Middle" />
                <textFieldExpression class="java.lang.String">
               <![CDATA[$F{name}]]>
                </textFieldExpression>
       </textField>
          </band>
     </detail>
</jasperReport>
```

下面的代码演示了上述jasper_report_template.jrxml文件的编译。

```
package com.yiibai;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;

public class JasperReportCompile {

   public static void main(String[] args) {
      String sourceFileName = "C://tools/jasperreports-5.0.1/test"
      "/jasper_report_template.jrxml";

      System.out.println("Compiling Report Design ...");
      try {
         /**
          * Compile the report to a file name same as
          * the JRXML file name
          */
         JasperCompileManager.compileReportToFile(sourceFileName);
      } catch (JRException e) {
         e.printStackTrace();
      }
      System.out.println("Done compiling!!! ...");
   }
}
```

## 模板编译

至于下一步，让我们保存上面的文件内容：**C: oolsjasperreports-5.0.1 estsrccomyiibaiJasperReportCompile.java** 并导入baseBuild.xml在为下面的build.xml文件。baseBuild.xml已经编译和运行的目标：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="run" basedir=".">

   <import file="baseBuild.xml"/>

</project>
```

接下来，让我们打开命令行窗口并转到build.xml文件放置的目录。最后执行的命令 **ant -Dmain-class=com.yiibai.JasperReportCompile** 如下:

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml
compile:
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:27:
    warning: 'includeantruntime' was not set, defaulting to
    build.sysclasspath=last;set to false for repeatable builds
    [javac] Compiling 1 source file to C:    oolsjasperreports-5.0

run:
     [echo] Runnin class : com.yiibai.JasperReportCompile
     [java] Compiling Report Design ...
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
     [java] log4j:WARN Please initialize the log4j system properly
     [java] Done compiling!!! ...

BUILD SUCCESSFUL
Total time: 8 seconds
```

正如上文编译的结果，会看到jasper_report_template.jasper得到了语言生成的模板文件在C: oolsjasperreports-5.0.1 est 目录。

## 预览编译报表模板

net.sf.jasperreports.view.JasperDesignViewer 正如在前面的章节中讨论可用来预览编译报告模板以及JRXML模板。

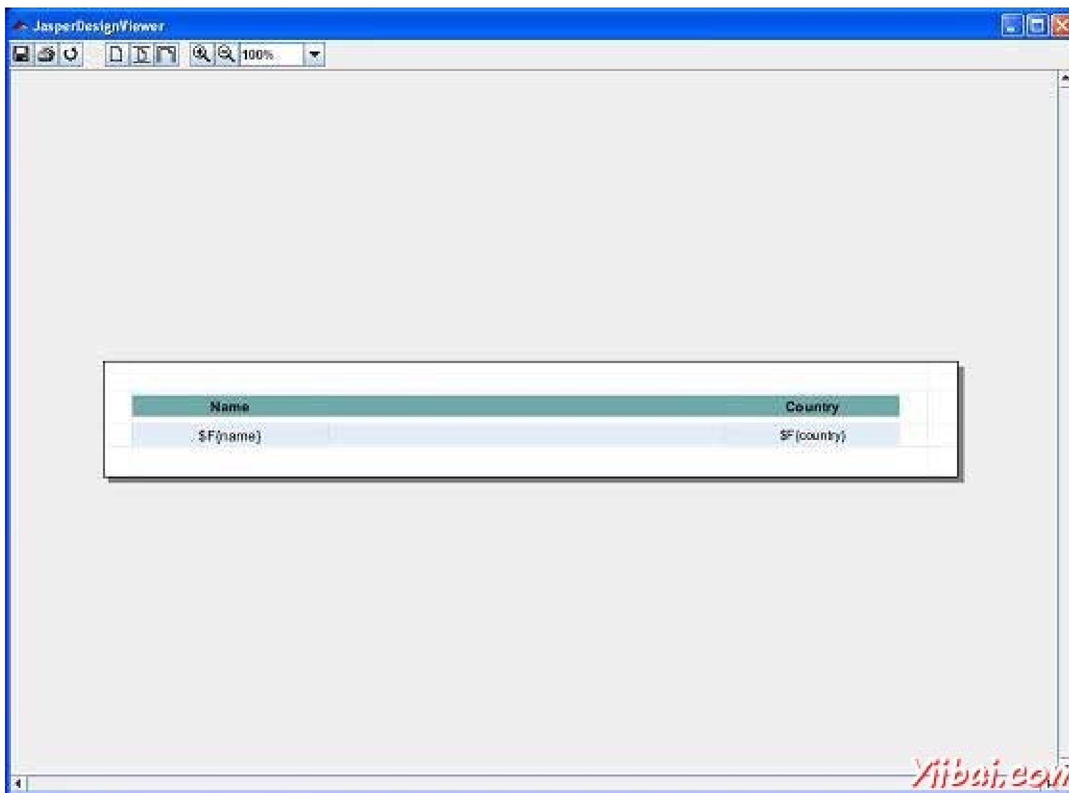
为了进一步推动，让我们添加一个新的目标viewDesign上述build.xml文件，这将让我们先看盾编译报告。下面是修改后build.xml： 导入文件 - baseBuild.xml做好环境设置，并应放置在同一目录中的build.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewDesign" basedir=".">

   <import file="baseBuild.xml" />
   <target name="viewDesign" description="Design viewer is launched
         to preview the compiled report design.">
      <java classname="net.sf.jasperreports.view.JasperDesignViewer"
            fork="true">
      <arg value="-F${file.name}.jasper" />
      <classpath refid="classpath" />
      </java>
   </target>

</project>
```

让我们执行命令：ant（viewDesign是默认的目标），在命令提示符下。JasperDesignViewer窗口打开显示Jasper文件如下：




## 通过ANT任务编译

报告模板编译更像是比一个运行时的工作设计时的工作，JasperReport库具有一个自定义ANT任务。因为当在运行时创建JRXML文件某些情况下，不能使用此ANT任务。自定义ANT任务被称为JRC和由类实现：net.sf.jasperreports.ant.JRAntCompileTask。其语法和行为是非常相似的内置 <javac> ANT任务。

## 模板编译

让我们添加新的目标编译报表设计，以我们现有的build.xml。这里使用与文件集的嵌套<src>标签中指定的源文件夹。嵌套的源标签允许编译器，可通过许多不同的地点分散，在一个单一的根报表源文件夹不进行分组报告模板。下面是修改后的build.xml：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="compilereportdesing" base
   <import file="baseBuild.xml" />
   <target name="viewDesign" description="Design viewer is launched
         to preview the compiled report design.">
      <java classname="net.sf.jasperreports.view.JasperDesignViewer
            fork="true">
      <arg value="-F${file.name}.jasper" />
      <classpath refid="classpath" />
      </java>
   </target>

   <target name="compilereportdesing" description="Compiles the JXM
         file and produces the .jasper file.">
      <taskdef name="jrc"
         classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
            <fileset dir=".">
            <include name="*.jrxml" />
            </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>

</project>
```

接下来，让我们打开命令提示符并转到build.xml文件放置的目录。执行命令ant（compilereportdesing是默认的目标）输出为如下：

```
C:    oolsjasperreports-5.0.1    est>ant
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly.
      [jrc] log4j:WARN See
      http://logging.apache.org/log4j/1.2/faq.htmll#noconfig
      for more info.
      [jrc] File :
      C:    oolsjasperreports-5.0.1    estjasper_report_template.jr

BUILD SUCCESSFUL
Total time: 5 seconds
```

文件jasper_report_template.jasper是在文件系统（在我们的例子中为 C: oolsjasperreports-5.0.1 est目录）产生的。这个文件是相同的通过调用 net.sf.jasperreports.engine.JasperCompileManager.compileReportToFile()程序生成一个。我们可以预览这个 jasper文件，执行ant viewDesign。

# JasperReport填充报表 - JasperReports教程

任何报告工具的主要目的是为了生产出高品质的文档。举报填充过程有助于报告工具通过操纵数据集来实现这一目标。需要报表填充过程的主要输入是：

- 报表模板：这是实际的JasperReport文件
- 报告参数：这些所传递的报表填充时间给引擎基本上都是命名的值。我们将在报表参数章节讨论。
- 数据源：我们可以从一系列像一个SQL查询，XML文件，CSV文件，一个HQL（Hibernate查询语言）查询，Java Beans的集合等数据源的填补Jasper这个文件将详细讨论在报表数据源的篇章。

这个过程产生的输出。jrprint是一个文档随时查看，打印或导出为其他格式。外观类net.sf.jasperreports.engine.JasperFillManager通常用于填充一个报表模板与数据。这个类有各种fillReportXXX()方法，填补报表模板（模板可以位于磁盘上，从输入流采集，或直接提供的内存）。

主要有两类在此外观类fillReportXXX()方法：

1. 第一种类型，接收java.sql.Connection对象作为第三个参数。大多数情况下报表都充满了从关系数据库中的数据。这是通过：
    - 通过JDBC连接到数据库。
    - 包括报表模板中的SQL查询。
    - JasperReports引擎使用传入的连接并执行SQL查询。
    - 因此，一个报表数据源产生填充的报告。
2. 第二类，收到net.sf.jasperreports.engine.JRDataSource对象，当提供其他形式的数据来填补。

## 填充报告模板

让我们来写一个报表模板。在JRXML文件(C: oolsjasperreports-5.0.1 estjasper_report_template.jrxml) 的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" language="groovy" pageWidth="595"
```

```
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">

    <queryString>
    <![CDATA[]]>
    </queryString>
    <field name="country" class="java.lang.String">
        <fieldDescription><![CDATA[country]]></fieldDescription>
    </field>
    <field name="name" class="java.lang.String">
        <fieldDescription><![CDATA[name]]></fieldDescription>
    </field>
    <columnHeader>
     <band height="23">
    <staticText>
        <reportElement mode="Opaque" x="0" y="3" width="535"
         height="15" backcolor="#70A9A9" />
        <box>
            <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
        </box>
        <textElement />
        <text><![CDATA[]]> </text>
    </staticText>
    <staticText>
        <reportElement x="414" y="3" width="121" height="15" />
        <textElement textAlignment="Center"
            verticalAlignment="Middle">
        <font isBold="true" />
        </textElement>
        <text><![CDATA[Country]]></text>
    </staticText>
    <staticText>
        <reportElement x="0" y="3" width="136" height="15" />
        <textElement textAlignment="Center"
            verticalAlignment="Middle">
        <font isBold="true" />
        </textElement>
        <text><![CDATA[Name]]></text>
    </staticText>
     </band>
    </columnHeader>
    <detail>
        <band height="16">
     <staticText>
         <reportElement mode="Opaque" x="0" y="0" width="535"
         height="14" backcolor="#E5ECF9" />
         <box>
                <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
         </box>
         <textElement />
         <text><![CDATA[]]> </text>
     </staticText>
     <textField>
```

```
            <reportElement x="414" y="0" width="121" height="15" />
            <textElement textAlignment="Center"
                verticalAlignment="Middle">
            <font size="9" />
            </textElement>
            <textFieldExpression class="java.lang.String">
                <![CDATA[$F{country}]]>
           </textFieldExpression>
       </textField>
       <textField>
           <reportElement x="0" y="0" width="136" height="15" />
           <textElement textAlignment="Center"
               verticalAlignment="Middle" />
               <textFieldExpression class="java.lang.String">
              <![CDATA[$F{name}]]>
               </textFieldExpression>
       </textField>
          </band>
      </detail>
  </jasperReport>
```

接下来，让我们通过Java数据对象（Java bean）集合，到Jasper报表引擎，填补了这一编译报告。

写一个POJO DataBean.java表示数据对象（Java bean）。这个类定义了两个字符串对象名称和国家。把它保存到目录 **C: oolsjasperreports-5.0.1 estsrccomyiibai**.

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

编写一个类DataBeanList.java具有业务逻辑生成java bean对象的集合。这是进一步传递到报表引擎，生成报表。在这里，我们添加在列表 DataBean进行对象。把它保存到目录**C: oolsjasperreports-5.0.1 estsrccomyiibai**.

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

编写一个类DataBeanList.java具有业务逻辑生成java bean对象的集合。这是进一步传递到Jasper报表引擎，生成报告。在这里，我们添加在列表4 DataBean进行对象。把它保存到目录 **C: oolsjasperreports-5.0.1 estsrccomyiibai**.

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
         "c://tools/jasperreports-5.0.1/test/jasper_report_template
      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      try {
         JasperFillManager.fillReportToFile(
         sourceFileName,
         parameters,
         beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

# 生成报表

现在，我们将编译并使用我们的定期Ant构建过程执行这些文件。 build.xml文件如下图所示：

> 导入文件 - baseBuild.xml环境设置，并应放置在同一目录中的build.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="executereport" basedir="
   <import file="baseBuild.xml"/>

   <target name="executereport" depends="compile,compilereportdesin
        <echo message="Im here"/>
   </target>
    <target name="compilereportdesing"
         description="Compiles the JXML file and
         produces the .jasper file.">
         <taskdef name="jrc"
         classname="net.sf.jasperreports.ant.JRAntCompileTask">
            <classpath refid="classpath" />
         </taskdef>
         <jrc destdir=".">
            <src>
            <fileset dir=".">
               <include name="*.jrxml" />
            </fileset>
            </src>
            <classpath refid="classpath" />
         </jrc>
   </target>
</project>
```

接下来，让我们打开命令行窗口并转到build.xml文件放置的目录。最后，执行命令 **ant -Dmain-class=com.yiibai.JasperReportFill** （executereport是默认的目标），如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

compile:
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:27:
    warning: 'includeantruntime' was not set, defaulting to
    build.sysclasspath=last; set to false for repeatable builds
    [javac] Compiling 1 source file to
    C:    oolsjasperreports-5.0.1    estclasses

run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly

BUILD SUCCESSFUL
Total time: 8 seconds
```

如上述执行结果的文件jasper_report_template.jrprint是在同一目录中为 .jasper文件（在这种情况下，它是产生在 C: oolsjasperreports-5.0.1 est).

# JasperReport查看和打印报告 - JasperReports教程

报表填充过程JasperPrint对象的输出可以使用内置的浏览器组件来查看，打印或导出到更多的流行的文件格式，如PDF，HTML，RTF，XLS，ODT，CSV或XML。Jasper文件查看和打印将包括在本章中。导出将包括在下一章导出报表.

## 查看报表

JasperReport提供了一个内置的浏览器观看原始格式生成的报表。这是一个基于Swing的组件和其他Java应用程序可以无需将文档导出为其他格式，以便查看或打印此集成组件。net.sf.jasperreports.view.JRViewer类表示这个可视组件。这个类也可以被定制为每个应用程序的需要，通过继承它。

JasperReports也有用来查看报表的可视化组件Swing应用程序。此应用程序可以帮助在相同的格式查看报表为*.jrprint就产生了。这个Swing应用程序是在类net.sf.jasperreports.view.JasperViewer实现。要使用此功能，我们可以把这个包成一个Ant目标，以查看报表。

### 查看生成的报告

下面的示例演示如何查看使用JasperViewer类的报表。

让我们来写一个报告模板。在JRXML文件(C: oolsjasperreports-5.0.1
estjasper_report_template.jrxml)的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">

    <queryString>
    <![CDATA[]]>
    </queryString>
    <field name="country" class="java.lang.String">
        <fieldDescription><![CDATA[country]]></fieldDescription>
    </field>
    <field name="name" class="java.lang.String">
        <fieldDescription><![CDATA[name]]></fieldDescription>
```

```
      </field>
      <columnHeader>
       <band height="23">
      <staticText>
          <reportElement mode="Opaque" x="0" y="3" width="535"
           height="15" backcolor="#70A9A9" />
          <box>
              <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
          </box>
          <textElement />
          <text><![CDATA[]]> </text>
      </staticText>
      <staticText>
          <reportElement x="414" y="3" width="121" height="15" />
          <textElement textAlignment="Center"
              verticalAlignment="Middle">
          <font isBold="true" />
          </textElement>
          <text><![CDATA[Country]]></text>
      </staticText>
      <staticText>
          <reportElement x="0" y="3" width="136" height="15" />
          <textElement textAlignment="Center"
              verticalAlignment="Middle">
          <font isBold="true" />
          </textElement>
          <text><![CDATA[Name]]></text>
      </staticText>
       </band>
      </columnHeader>
      <detail>
          <band height="16">
       <staticText>
           <reportElement mode="Opaque" x="0" y="0" width="535"
           height="14" backcolor="#E5ECF9" />
           <box>
                  <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
           </box>
           <textElement />
           <text><![CDATA[]]> </text>
       </staticText>
       <textField>
           <reportElement x="414" y="0" width="121" height="15" />
           <textElement textAlignment="Center"
               verticalAlignment="Middle">
           <font size="9" />
           </textElement>
           <textFieldExpression class="java.lang.String">
               <![CDATA[$F{country}]]>
          </textFieldExpression>
       </textField>
       <textField>
           <reportElement x="0" y="0" width="136" height="15" />
```

```
          <textElement textAlignment="Center"
              verticalAlignment="Middle" />
              <textFieldExpression class="java.lang.String">
             <![CDATA[$F{name}]]>
              </textFieldExpression>
      </textField>
         </band>
     </detail>
 </jasperReport>
```

接下来，让我们通过Java数据对象（Java bean）的集合，到Jasper报表引擎，填补了这一编译报告。

写一个POJO DataBean.java表示数据对象（的Java bean）。这个类定义了两个字符串对象name和country。把它保存到目录 **C: oolsjasperreports-5.0.1 estsrccomyiibai**.

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

编写一个类DataBeanList.java具有业务逻辑生成java bean对象的集合。这是进一步传递到Jasper 报表引擎，生成报告。在这里，我们添加在列表中的4个DataBean进行对象。把它保存到目录**C: oolsjasperreports-5.0.1 estsrccomyiibai**.

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

写一个主类文件JasperReportFill.java，它从类（DataBeanList）得到的java bean的集合，并将其传递到Jasper报表引擎，填补了报告模板。把它保存到目录 C:oolsjasperreports-5.0.1 estsrccomyiibai.

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
         "c://tools/jasperreports-5.0.1/test/jasper_report_template
      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      try {
         JasperFillManager.fillReportToFile(
         sourceFileName,
         parameters,
         beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

让我们来写一个目标viewFillReport的build.xml文件。 build.xml文件如下所示：

> 导入文件 - baseBuild.xml环境设置，并应放置在同一目录中的build.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir='
   <import file="baseBuild.xml"/>

   <target name="viewFillReport" depends="compile,compilereportdesi
      description="Launches the report viewer
      to preview the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer" fork
         <arg value="-F${file.name}.JRprint" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>
</project>
```

接下来，让我们打开命令行窗口并转到build.xml文件放置的目录。最后执行的命令 **ant -Dmain-class=com.yiibai.JasperReportFill**（viewFillReport是默认的目标）。因此，我们看到一个JasperViewer窗口，如下面的屏幕：

# 打印报表

我们可以使用net.sf.jasperreports.engine.JasperPrintManager类打印的JasperReports类库生成的文件（在他们的专有格式i.eJasperPrint对象）。这是依赖于Java2 API打印一个假象类。我们还可以打印文档，一旦JasperReport的文档导出为其他格式，如HTML或PDF。

## 打印生成的报告

下面的代码演示报表的打印。让我们更新现有的类JasperReportFill。我们将使用JasperPrintManager.printReport()方法。此方法需要源文件名.jrprint（这里我们通过我们在上一步生成的使用方法JasperFillManager.fillReportToFile()）作为第一个参数。第二个参数是布尔值，用于显示标准打印对话框（我们将其设置为true这里）。

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrintManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName = "c://tools/jasperreports-5.0.1/" +
              "test/jasper_report_template.jasper";
      String printFileName = null;
      DataBeanList DataBeanList = new DataBeanList();
      ArrayList <databean>dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      try {
          printFileName = JasperFillManager.fillReportToFile(
         sourceFileName,
         parameters,
         beanColDataSource);
         if(printFileName != null){
           JasperPrintManager.printReport(
               printFileName, true);
      }
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}</databean>
```

现在，让我们将此文件保存到目录C: oolsjasperreports-5.0.1 estsrccomyiibai. 我们将使用ANT编译并执行此文件.build.xml文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="executereport" basedir=".
   <import file="baseBuild.xml"/>

   <target name="executereport"
   depends="compile,compilereportdesing,run">
        <echo message="Im here"/>
   </target>
    <target name="compilereportdesing"
         description="Compiles the JXML file and
         produces the .jasper file.">
         <taskdef name="jrc"
         classname="net.sf.jasperreports.ant.JRAntCompileTask">
            <classpath refid="classpath" />
         </taskdef>
         <jrc destdir=".">
            <src>
            <fileset dir=".">
               <include name="*.jrxml" />
            </fileset>
            </src>
            <classpath refid="classpath" />
         </jrc>
   </target>
</project>
```

接下来，让我们打开命令提示符并转到build.xml文件放置的目录。最后，执行命令 **ant -Dmain-class=com.yiibai.JasperReportPrint**. 因此，会出现一个打印对话框。单击确定以打印文档。

# JasperReport导出报表 - JasperReports教程

我们已经看到在前面的章节中，如何打印和查看的JasperReport生成的文档。在这里，我们将看到如何在其他格式，如PDF，HTML和XLS转换或导出这些报告。Facade类net.sf.jasperreports.engine.JasperExportManager提供实现这一功能。导出方式转变JasperPrint对象（.jrprint文件）导入到不同的格式。

下面的代码（JasperReportExport.java）演示了JasperReport文档的导出过程。该JasperExportManager提供的方法将报表导出成PDF，HTML和XML。导出到使用的类net.sf.jasperreports.engine.export.JRXlsExporter的XLS格式。此代码生成以下三个文件：

- sample_report.pdf
- sample_report.htmll
- sample_report.xls

## 导出为其他格式

让我们来写一个报表模板。在JRXML文件的内容 (C: oolsjasperreports-5.0.1 estjasper_report_template.jrxml) 如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">

    <queryString>
    <![CDATA[]]>
    </queryString>
    <field name="country" class="java.lang.String">
        <fieldDescription><![CDATA[country]]></fieldDescription>
    </field>
    <field name="name" class="java.lang.String">
        <fieldDescription><![CDATA[name]]></fieldDescription>
    </field>
    <columnHeader>
     <band height="23">
    <staticText>
        <reportElement mode="Opaque" x="0" y="3" width="535"
         height="15" backcolor="#70A9A9" />
```

```
        <box>
            <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
        </box>
        <textElement />
        <text><![CDATA[]]> </text>
    </staticText>
    <staticText>
        <reportElement x="414" y="3" width="121" height="15" />
        <textElement textAlignment="Center"
            verticalAlignment="Middle">
        <font isBold="true" />
        </textElement>
        <text><![CDATA[Country]]></text>
    </staticText>
    <staticText>
        <reportElement x="0" y="3" width="136" height="15" />
        <textElement textAlignment="Center"
            verticalAlignment="Middle">
        <font isBold="true" />
        </textElement>
        <text><![CDATA[Name]]></text>
    </staticText>
     </band>
    </columnHeader>
    <detail>
        <band height="16">
     <staticText>
         <reportElement mode="Opaque" x="0" y="0" width="535"
         height="14" backcolor="#E5ECF9" />
         <box>
                <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
         </box>
         <textElement />
         <text><![CDATA[]]> </text>
     </staticText>
     <textField>
         <reportElement x="414" y="0" width="121" height="15" />
         <textElement textAlignment="Center"
             verticalAlignment="Middle">
         <font size="9" />
         </textElement>
         <textFieldExpression class="java.lang.String">
             <![CDATA[$F{country}]]>
        </textFieldExpression>
     </textField>
     <textField>
         <reportElement x="0" y="0" width="136" height="15" />
         <textElement textAlignment="Center"
             verticalAlignment="Middle" />
             <textFieldExpression class="java.lang.String">
            <![CDATA[$F{name}]]>
             </textFieldExpression>
     </textField>
```

```
        </band>
    </detail>
</jasperReport>
```

接着，POJO文件 C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBean.java 的内容如下：

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

文件 C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBeanList.java 的内容如下：

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

写一个主类文件JasperReportFill.java，它从类（DataBeanList）获取java bean集合，并将其传递到Jasper 报表引擎，填补了报表模板。把它保存到目录 **C:oolsjasperreports-5.0.1 estsrccomyiibai**.

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JRExporterParameter;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;
import net.sf.jasperreports.engine.export.JRXlsExporter;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName = "c://tools/jasperreports-5.0.1/"
```

```
            + "test/jasper_report_template.jasper";
      String printFileName = null;
      DataBeanList DataBeanList = new DataBeanList();
      ArrayList <databean>dataList = DataBeanList.getDataBeanList()
      JRBeanCollectionDataSource beanColDataSource =
         new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      try {
         printFileName = JasperFillManager.fillReportToFile(sourceF
            parameters, beanColDataSource);
         if (printFileName != null) {
            /**
             * 1- export to PDF
             */
            JasperExportManager.exportReportToPdfFile(printFileName
                  "C://sample_report.pdf");

            /**
             * 2- export to HTML
             */
            JasperExportManager.exportReportToHtmlFile(printFileNam
                  "C://sample_report.htmll");

            /**
             * 3- export to Excel sheet
             */
            JRXlsExporter exporter = new JRXlsExporter();

            exporter.setParameter(JRExporterParameter.INPUT_FILE_NA
                  printFileName);
            exporter.setParameter(JRExporterParameter.OUTPUT_FILE_N
                  "C://sample_report.xls");

            exporter.exportReport();
         }
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}</databean>
```

在这里，我们已经包括了逻辑Jasper 打印文件导出到PDF，HTML和XLS格式。

## 生成报表

让我们用我们的常规Ant构建过程编译和执行上述文件。 build.xml文件如下图所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="executereport" basedir="
   <import file="baseBuild.xml"/>

   <target name="executereport" depends="compile,compilereportdesir
        <echo message="Im here"/>
   </target>
    <target name="compilereportdesing"
         description="Compiles the JXML file and
         produces the .jasper file.">
         <taskdef name="jrc"
         classname="net.sf.jasperreports.ant.JRAntCompileTask">
            <classpath refid="classpath" />
         </taskdef>
         <jrc destdir=".">
            <src>
            <fileset dir=".">
               <include name="*.jrxml" />
            </fileset>
            </src>
            <classpath refid="classpath" />
         </jrc>
   </target>
</project>
```

进入命令提示符，然后转到目录C: oolsjasperreports-5.0.1 est，其中build.xml已放置。最后执行的命令 ant -Dmain-class=com.yiibai.JasperReportFill. 输出如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

clean-sample:
   [delete] Deleting directory C:    oolsjasperreports-5.0.1    est
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re

compile:
    [mkdir] Created dir: C:    oolsjasperreports-5.0.1    estclasse
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:28:
    warning: 'includeantruntime' was not set, defaulting t
    [javac] Compiling 4 source files to C:    oolsjasperreports-5.0

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq
      [jrc] File : C:    oolsjasperreports-5.0.1    estjasper_repor

run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly

executereport:
     [echo] Im here

BUILD SUCCESSFUL
Total time: 32 seconds
```

正如上面执行的结果，会发现三个文件sample_report.pdf，sample_report.htmll, sample_report.xls已生成在 C: 目录.

# JasperReport报表参数 - JasperReports教程

填充一个报表的主要输入是：报表模板，参数和数据源。本章将介绍这些参数，并在接下来的章节中，我们将介绍数据源。

参数是在报表填充操作传递给报表引擎的对象引用。参数传递有用的数据到报表引擎，它可以不通过数据源来传递的数据是有用的。如作者姓名，报告等的标题数据，可以通过参数传递。 Jasper报表模板或JRXML模板可以包含零个或多个参数的元素。

## 参数声明

参数声明很简单，如下所示：

```
<parameter name="exampleParameter" class="java.lang.String" />
```

### name属性

parameter>元素的name属性是强制性的。它通过名称引用的参数在报表表达式。参数名应该是一个单词。它不应该包含任何特殊字符，如句号或逗号。

### class属性

class属性也是强制性的，它指定了参数值的类名。它的默认值是java.lang.String。这是可以改变的，以在运行时可用任何类。不论报表参数的类型，引擎采用构造于$P{}标记是用来报表表达，从而使手工投射不必要。

## 内置参数

以下是预定义的报表参数，准备在表达式中使用：

| Parameter Name | 描述 |
|---|---|
| REPORT_PARAMETERS_MAP | 包含所有用户定义和内置参数映射 |
| REPORT_CONNECTION | 这指向用于JDBC数据源的用户提供java.sql.Connection中 |
| REPORT_DATA_SOURCE | 这是JRDataSource代表任一用户提供的实例的内置的数据源类型或用户定义 |
| REPORT_MAX_COUNT | 这是一个java.lang.Integer的值，从而允许用从数据源限制记录。 |
| REPORT_SCRIPTLET | 这指向net.sf.jasperreports.engine.JRAbstractScrip和包含报表的scriptlet，由用户提供的一个实 |
| REPORT_LOCALE | 这是一个java.util.Locale的实例，包含资源包需的语言环境 |
| REPORT_RESOURCE_BUNDLE | 这指向java.util.ResourceBundle对象和包含地化的消息 |
| REPORT_TIME_ZONE | 这是一个java.util.TimeZone的实例，用于日格式 |
| REPORT_VIRTUALIZER | 这是net.sf.jasperreports.engine.JRVirtualizerobj一个实例，以及用于网页的虚拟化（优化内消耗） |
| REPORT_CLASS_LOADER | 这是在报告充填过程中使用的加载，如图像字体和子报表模板资源java.lang.ClassLoad的实例 |
| IS_IGNORE_PAGINATION | 如果设置为java.lang.Boolean.TRUE报告将一个很长的网页和分页符来产生不会发生 |

# 例子

让我们通过ReportTitle和 Author报表（由JasperReportFill.java生成）。经修订的文件C: oolsjasperreports-5.0.1 estsrccomyiibaiJasperReportFill.java 如下：

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
      "C://tools/jasperreports-5.0.1/test/jasper_report_template.ja

      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

      try {
         JasperFillManager.fillReportToFile(
         sourceFileName, parameters, beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

POJO文件的内容 C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBean.java 如下所示：

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

文件 **C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBeanList.java** 的内容如下：

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

让我们添加参数<ReportTitle>和<AUTHOR>我们现有的报告模板（章报表设计）。报告标题和作者将在报表的开头显示。修订后的报告模板（jasper_report_template.jrxml）如下。将其保存到 C: oolsjasperreports-5.0.1 est 目录:

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC
"//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" pageWidth="595"
pageHeight="842" columnWidth="515"
leftMargin="40" rightMargin="40" topMargin="50" bottomMargin="50">
<parameter name="Author" class="java.lang.String"/>**

   <queryString>
    <![CDATA[]]>
```

```
    </queryString>

    <field name="country" class="java.lang.String">
       <fieldDescription><![CDATA[country]]></fieldDescription>
    </field>

    <field name="name" class="java.lang.String">
       <fieldDescription><![CDATA[name]]></fieldDescription>
    </field>

    <title>
       <band height="70">
          <line>
             <reportElement x="0" y="0" width="515"
             height="1"/>
          </line>
          <textField isBlankWhenNull="true" bookmarkLevel="1">
             <reportElement x="0" y="10" width="515"
             height="30"/>
             <textElement textAlignment="Center">
             <font size="22"/>
             </textElement>
             <textFieldExpression class="java.lang.String">
             <![CDATA[$P{ReportTitle}]]>
             </textFieldExpression>
             <anchorNameExpression><![CDATA["Title"]]>
             </anchorNameExpression>
             </textField>
             <textField isBlankWhenNull="true">
             <reportElement  x="0" y="40" width="515" height="20"/>
             <textElement textAlignment="Center">
                  <font size="10"/>
             </textElement>
             <textFieldExpression class="java.lang.String">
             <![CDATA[$P{Author}]]>
             </textFieldExpression>
             </textField>
       </band>
    </title>

    <columnHeader>
       <band height="23">
          <staticText>
             <reportElement mode="Opaque" x="0" y="3"
             width="535"    height="15"
             backcolor="#70A9A9" />
             <box>
             <bottomPen lineWidth="1.0"
             lineColor="#CCCCCC" />
             </box>
             <textElement />
             <text><![CDATA[]]>
             </text>
```

```
         </staticText>
         <staticText>
            <reportElement x="414" y="3" width="121"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
                   <font isBold="true" />
            </textElement>
            <text><![CDATA[Country]]></text>
         </staticText>
         <staticText>
            <reportElement x="0" y="3" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font isBold="true" />
            </textElement>
            <text><![CDATA[Name]]></text>
         </staticText>
      </band>
   </columnHeader>

   <detail>
      <band height="16">
         <staticText>
            <reportElement mode="Opaque" x="0" y="0"
            width="535"    height="14"
            backcolor="#E5ECF9" />
            <box>
               <bottomPen lineWidth="0.25"
               lineColor="#CCCCCC" />
            </box>
            <textElement />
            <text><![CDATA[]]>
            </text>
         </staticText>
         <textField>
            <reportElement x="414" y="0" width="121"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font size="9" />
            </textElement>
            <textFieldExpression class="java.lang.String">
            <![CDATA[$F{country}]]>
            </textFieldExpression>
         </textField>
         <textField>
            <reportElement x="0" y="0" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle" />
            <textFieldExpression class="java.lang.String">
```

```
                <![CDATA[$F{name}]]>
                </textFieldExpression>
            </textField>
        </band>
    </detail>
</jasperReport>
```

# 报表生成

编译和执行使用常规Ant构建过程上面的文件。 build.xml文件中的内容（根据目录保存C: oolsjasperreports-5.0.1 est）情况如下。

> 导入文件 - baseBuild.xml可以从环境设置章节中了解，并应放置在同一目录中的build.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir='
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>
</project>
```

接下来，让我们打开命令行窗口并转到build.xml文件放置的目录。最后执行的命令 **ant -Dmain-class=com.yiibai.JasperReportFill** （viewFullReport是默认的目标），如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

clean-sample:
   [delete] Deleting directory C:    oolsjasperreports-5.0.1    est
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re

compile:
    [mkdir] Created dir: C:    oolsjasperreports-5.0.1    estclasse
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:28: wa
    'includeantruntime' was not set, defaulting to build.sysclasspa
    set to false for repeatable builds
    [javac] Compiling 7 source files to C:    oolsjasperreports-5.0

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq
      for more info.
      [jrc] File : C:    oolsjasperreports-5.0.1    estjasper_repor

run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly

BUILD SUCCESSFUL
Total time: 18 seconds
```

正如上文编译的结果，JasperViewer窗口打开如下面的屏幕：

在这里，我们看到的是，报表头 "List Of Contacts"，并着有"Prepared By Manisha" 显示在报表的开头。

# JasperReports报表数据源 - JasperReports教程

数据源的结构数据容器。同时生成报告，Jasper报表引擎获得来自数据源的数据。数据可以从数据库，XML文件，对象数组和集合中的对象来获得。我们将在本章填充报告所看到的fillReportXXX()方法，预计将收到该报告的数据源其以填充，在net.sf.jasperreports.engine.JRDataSource对象或一个java.sql.Connection中的形式（当报表数据在关系数据库中找到）。

JRDataSource接口只有两个方法，这应该被实现：

1. public boolean next() throws JRException;

2. public Object getFieldValue(JRField jrField) throws JRException;

从数据源中检索数据的唯一方法是通过使用报表字段。有一个JRDataSource接口的几个默认的实现，根据不同的数据源中的记录被获得的方式。

## 数据源的实现

下表总结了数据源和它们的实现类。

| Datasource | Implementation Class |
|---|---|
| JDBC | net.sf.jasperreports.engine.JRResultSetDataSource |
| JavaBean | net.sf.jasperreports.engine.data.JRBeanCollectionDataSource, net.sf.jasperreports.engine.data.JRBeanArrayDataSource |
| Map-based | net.sf.jasperreports.engine.data.JRMapArrayDataSource, net.sf.jasperreports.engine.data.JRMapCollectionDataSource |
| TableModel | net.sf.jasperreports.engine.data.JRTableModelDataSource |
| XML | net.sf.jasperreports.engine.data.JRXmlDataSource |
| CSV | net.sf.jasperreports.engine.data.JRCsvDataSource |
| XLS | net.sf.jasperreports.engine.data.JRXlsDataSource |
| Empty | net.sf.jasperreports.engine.JREmptyDataSource |

### JDBC数据源

JRResultSetDataSource类关联入一个java.sql.ResultSet对象。这是当报表数据从关系数据库中提取最常用的数据源实现。如果ajava.sql.Connection传递给引擎来代替，它首先执行相关的查询，并将该返回java.sql.ResultSet中的对象在一个JRResultSetDataSource实例。

## JAVABEAN数据来源

JRBeanArrayDataSource类和JRBeanCollectionDataSource表示实现，可以分别包装的JavaBean对象的数组或集合。数组或集合中的每个对象都将被视为对这种类型的数据源中的一个记录。一个特定的JavaBean属性和相应的报表字段之间的映射是通过命名约定进行。报表字段的名称必须是相同的所指定的JavaBeans的规范JavaBean属性的名称。

> 在本教程中的所有例子中，我们使用JRBeanCollectionDataSource。

## 基于MAP的数据来源

如果父级应用程序已经存储在内存中的java.util.Map对象提供的申报数据的实现类JRMapArrayDataSource和JRMapCollectionDataSource非常有用。被包装的数组或集合中的每个映射对象被认为是数据源中的一个虚拟的记录，每个报表字段的值从映射中使用报表字段名作为键提取。

## TableModel的数据来源

在许多客户端应用程序，数据以表格形式显示。在许多应用中常见的需求是允许用户打印该表格形式的报告。实现类JRTableModelDataSource使生成的表格格式的Swing应用程序报告的任务。这个类封装了一个javax.swing.table.TableModel对象。列在包装的TableModel对象可以通过他们的名字或他们的基于0索引来访问。

## XML数据源

类JRXmlDataSource是基于DOM，它使用XPath表达式来选择XML文档数据的数据源的实现。 XML数据源中的记录是通过XPath表达式选择的节点元素表示。字段值是由每个记录使用由字段描述（JRXML<fieldDescription>元素）所提供的XPath表达式检索。

> XPath是用于导航XML文档的属性和元素的语言。有关XPath更多信息可以在这里找到http://www.w3.org/TR/xpath.

## CSV数据来源

JRCsvDataSource 代表了从结构化文本文件中检索其数据的数据源的实现，通常为CSV。字段值是正在使用他们的列索引检索。

## xls数据来源

JRXlsDataSource 代表其检索的Excel文件的数据的数据源的实现。报表字段映射为这个数据源的实现也是基于字段列索引。

## 空数据来源

类JREmptyDataSource，模拟与内部虚拟空的记录给定数量的数据源。它是由用户界面的工具来提供基本的报表预览功能，或在特殊报告模板，或用于测试和调试目的。

# 重绕数据源

net.sf.jasperreports.engine.JRRewindableDataSource扩展的基本JRDataSourceinterface。它增加了只有一个方法为MoveFirst()到接口。这种方法的目的是将光标移动到数据源中的第一个元素。

与放置在带内子报表不允许拆分由于isSplitAllowed="false"的设定，并且没有足够的空间，在当前页上对要呈现的子报表工作时，重绕的数据源是有用的。

以上所有数据源的实现是可回退除JRResultSetDataSource，因为它不支持移动记录指针回来。这对只有当该数据源是用它传递给子报表之前手动换一个java.sql.ResultSet中的一个问题。这是没有问题，如果SQL查询驻留在子报表模板，该引擎将在下一个页面上重新启动子报表时，再次执行它。

# 数据源提供者

JasperReports库有一个接口net.sf.jasperreports.engine.JRDataSourceProvider。这有助于创建和处理数据源对象。当创建使用GUI工具报表模板，则需要自定义报表的数据源的特殊工具。JRDataSourceProvider是为了堵塞自定义数据源到设计工具的标准方法。自定义实现该接口应实现以下方法，使创建和配置数据源对象和方法，上面列出数据源如果可能的话，里面可用的报表字段：

```
public boolean supportsGetFieldsOperation();

 public JRField[] getFields(JasperReport report)
     throws JRException, UnsupportedOperationException;

 public JRDataSource create(JasperReport report) throws JRExceptio

 public void dispose(JRDataSource dataSource) throws JRException;
```

# JasperReports报表字段 - JasperReports教程

报表字段是代表数据源和报表模板之间的数据映射元素。字段可以在报告中的表达式进行组合，以获得所需的输出。报表模板可以包含零个或更多的<field>元素。当声明报表字段，数据源应提供相应的数据到所有在报告模板中定义的字段。

## 字段声明

字段声明做如下：

```
<field name="FieldName" class="java.lang.String"/>
```

### name属性

<field>元素的name属性是强制性的。它通过名称引用的报表表达的字段。

### class属性

class属性指定的字段值的类名。它的默认值是java.lang.String。这是可以改变的，以在运行时可用任何类。不论一个报表字段的类型，引擎采用铸造于该$ F{}标记用于报表表达式，从而使手工投射不必要。

## 字段描述

<fieldDesciption>元素是可选元素。实现自定义的数据源，例如，当这是非常有用的。我们可以存储一个密钥或一些信息，使用它我们可以在运行时自定义数据源中检索字段的值。通过使用<fieldDesciption>元素而不是字段名，可以检索从数据源中的字段值时容易克服字段命名约定的限制。

下面是一段代码从我们现有的jrxml文件（第二章报告的设计）。在这里我们可以看到name, class 和 fieldDescription 元素的使用。

```
<field name="country" class="java.lang.String">
    <fieldDescription><![CDATA[country]]></fieldDescription>
</field>
<field name="name" class="java.lang.String">
    <fieldDescription><![CDATA[name]]></fieldDescription>
</field>
```

## 排序字段

数据的排序是必要的，数据源实现不支持它（如CSV数据源），JasperReports的支持在内存领域为基础的数据源排序。排序可以使用的报表模板的一个或多个<sortField>元素来完成。

如果指定ATLEAST1排序字段，在报告填充过程中的数据源传递给JRSortableDataSource实例。这反过来从获取数据源中的所有记录，根据指定的字段进行排序的内存中，并替换原来的数据源。

排序字段名称应该是相同的报表字段名称。用于排序的字段应该具有实现java.util.Comparable类型。进行自然顺序排序的所有字段除外java.lang.String类型（对于字符串类型，对应报告的填充区域自动分页使用）。当有多个排序字段指定，排序将使用的字段进行中它们出现在报告模板的顺序排序键。下面的例子demonstartes显示排序功能。

# Sorted Report 例子

sortField>元素添加到我们现有的报告模板（第报表设计）。让我们排序字段country 降序排列。修订后的报告模板（jasper_report_template.jrxml）如下。将其保存到C: oolsjasperreports-5.0.1 est directory:

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC
"//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
<parameter name="ReportTitle" class="java.lang.String"/>
<parameter name="Author" class="java.lang.String"/>
   <queryString>
    <![CDATA[]]>
   </queryString>
   <field name="country" class="java.lang.String">
      <fieldDescription><![CDATA[country]]></fieldDescription>
   </field>
   <field name="name" class="java.lang.String">
      <fieldDescription><![CDATA[name]]></fieldDescription>
   </field>
  **<sortField name="country" order="Descending"/>
   <sortField name="name"/>**
   <title>
      <band height="70">
         <line>
            <reportElement x="0" y="0" width="515"
            height="1"/>
         </line>
         <textField isBlankWhenNull="true" bookmarkLevel="1">
            <reportElement x="0" y="10" width="515"
            height="30"/>
            <textElement textAlignment="Center">
            <font size="22"/>
```

```
              </textElement>
              <textFieldExpression class="java.lang.String">
              <![CDATA[$P{ReportTitle}]]>
              </textFieldExpression>
              <anchorNameExpression><![CDATA["Title"]]>
              </anchorNameExpression>
              </textField>
              <textField isBlankWhenNull="true">
              <reportElement  x="0" y="40" width="515" height="20"/>
              <textElement textAlignment="Center">
                   <font size="10"/>
              </textElement>
              <textFieldExpression class="java.lang.String">
              <![CDATA[$P{Author}]]>
              </textFieldExpression>
              </textField>
        </band>
     </title>
     <columnHeader>
        <band height="23">
           <staticText>
              <reportElement mode="Opaque" x="0" y="3"
              width="535"    height="15"
              backcolor="#70A9A9" />
              <box>
              <bottomPen lineWidth="1.0"
              lineColor="#CCCCCC" />
              </box>
              <textElement />
              <text><![CDATA[]]>
              </text>
           </staticText>
           <staticText>
              <reportElement x="414" y="3" width="121"
              height="15" />
              <textElement textAlignment="Center"
              verticalAlignment="Middle">
                     <font isBold="true" />
              </textElement>
              <text><![CDATA[Country]]></text>
           </staticText>
           <staticText>
              <reportElement x="0" y="3" width="136"
              height="15" />
              <textElement textAlignment="Center"
              verticalAlignment="Middle">
                 <font isBold="true" />
              </textElement>
              <text><![CDATA[Name]]></text>
           </staticText>
        </band>
         </columnHeader>
         <detail>
```

```
         <band height="16">
            <staticText>
               <reportElement mode="Opaque" x="0" y="0"
               width="535"    height="14"
               backcolor="#E5ECF9" />
               <box>
                  <bottomPen lineWidth="0.25"
                  lineColor="#CCCCCC" />
               </box>
               <textElement />
               <text><![CDATA[]]>
               </text>
            </staticText>
            <textField>
               <reportElement x="414" y="0" width="121"
               height="15" />
               <textElement textAlignment="Center"
               verticalAlignment="Middle">
                  <font size="9" />
               </textElement>
               <textFieldExpression class="java.lang.String">
               <![CDATA[$F{country}]]>
               </textFieldExpression>
            </textField>
            <textField>
               <reportElement x="0" y="0" width="136"
               height="15" />
               <textElement textAlignment="Center"
               verticalAlignment="Middle" />
               <textFieldExpression class="java.lang.String">
               <![CDATA[$F{name}]]>
               </textFieldExpression>
            </textField>
         </band>
      </detail>
   </jasperReport>
```

在java代码报告填充保持不变。该文件 C: oolsjasperreports-5.0.1 estsrccomyiibaiJasperReportFill.java 的内容如下：

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
      "C://tools/jasperreports-5.0.1/test/jasper_report_template.ja

      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

      try {
         JasperFillManager.fillReportToFile(
         sourceFileName, parameters, beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

POJO文件**C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBean.java**的内容如下：

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

文件 **C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBeanList.java** 的内容如下：

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

# 报表生成

我们将编译和执行使用我们常规Ant构建过程上面的文件。 build.xml文件（目录下保存的内容 C: oolsjasperreports-5.0.1 est) 如下：

> 导入文件 - baseBuild.xml从环境设置章节中了解，并应放置在同一目录中的build.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir='
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>
</project>
```

接下来，打开命令行窗口并转到build.xml文件放置的目录。最后执行命令 **ant -Dmain-class=com.yiibai.JasperReportFill** （viewFullReport是默认的目标），如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

clean-sample:
   [delete] Deleting directory C:    oolsjasperreports-5.0.1    est
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re

compile:
    [mkdir] Created dir: C:    oolsjasperreports-5.0.1    estclasse
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:28: wa
    'includeantruntime' was not set, defaulting to build.sysclasspa
    set to false for repeatable builds
    [javac] Compiling 7 source files to C:    oolsjasperreports-5.0

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly.
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq.
      for more info.
      [jrc] File : C:    oolsjasperreports-5.0.1    estjasper_repor

run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 18 seconds
```

正如上文编译的结果，一个JasperViewer窗口打开如下面的屏幕：




在这里，我们可以看到，country排列的字母顺序降序排列。

# JasperReports报表表达式 - JasperReports教程

报表表达式是JasperReports使我们能够显示在报表上的数据计算的强大功能。计算出数据不是一个静态数据，并且不受特别的报表参数或数据源字段传递的数据。报表表达式是由组合报表参数，字段和静态数据。默认情况下，Java语言是用于编写报表的表达式。其他脚本语言如Groovy脚本语言，JavaScript或BeanShell脚本，报表表达式是由JasperReports编译器支持。

本章将解释如何报表表达式工作假设他们一直只用Java语言编写的。在JRXML报表模板，那里有定义表达式几个元素，如下所示：

- <variableExpression>
- <initialValueExpression>
- <groupExpression>
- <printWhenExpression>
- <imageExpression>
- <textFieldExpression>

## 声明表达式

基本上，所有的报表表达式是可以参考的报表字段，报表变量和报表参数Java表达式。

### 字段引用表达式

使用在表达式中一个报表字段参考，字段的名称必须放在$F{ 和 }字符序列之间，如下图所示。

```
<textfieldexpression>
   $F{Name}
</textfieldexpression>
```

下面是一段代码从我们现有的jrxml文件，从报表设计 章节中了解：

```
<textFieldExpression class="java.lang.String">
    <![CDATA[$F{country}]]>
</textFieldExpression>
```

### 变量引用表达式

引用在表达式中的变量，我们必须把像在下面的例子中的变量名放在$V {和}之间：

```
<textfieldexpression>
   "Total height : " + $V{SumOfHeight} + " ft."
</textfieldexpression>
```

## 参数参考表达

引用在表达式中的一个参数，该参数的名称应放在$ P{和}之间，如下面的例子：

```
<textfieldexpression>
   "ReportTitle : " + $P{Title}
</textfieldexpression>
```

下面是一段代码从现有的jrxml文件，这用来表示参数在表达式中引用。

```
<textField isBlankWhenNull="true" bookmarkLevel="1">
    <reportElement x="0" y="10" width="515" height="30"/>
    <textElement textAlignment="Center">
        <font size="22"/>
    </textElement>
    <textFieldExpression class="java.lang.String">
       <![CDATA[$P{ReportTitle}]]>
    </textFieldExpression>
    <anchorNameExpression>
       <![CDATA["Title"]]>
    </anchorNameExpression>
</textField>
<textField isBlankWhenNull="true">
    <reportElement  x="0" y="40" width="515" height="20"/>
    <textElement textAlignment="Center">
        <font size="10"/>
    </textElement>
    <textFieldExpression class="java.lang.String">
       <![CDATA[$P{Author}]]>
    </textFieldExpression>
</textField>
```

正如在上面看到，参数，字段和变量引用，其实是真正的Java对象。从参数，字段或在报表模板所作的变量声明知道他们的类，甚至可以在表达式中调用的对象引用的方法。

下面的示例演示如何提取并显示java.lang.String报表字段的第一个字母 "Name":

```
<textFieldExpression>
    $F{Name}.substring(0, 1)
</textFieldExpression>
```

### 资源包参考表达

引用在表达式中的资源，关键要$R{和}之间放像下面的例子：

```
<textfieldexpression>
   $R{report.title}
</textfieldexpression>
```

基于运行时提供的语言环境和report.title键，报表模板相关的资源包加载。因此，报表标题是从资源包中提取字符串值显示。更多关于国际化可以在国际化一章中找到。

## 计算器

计算器是JasperReports，其计算表达式和增量变量或数据集在报表填充时间的实体。在编译过程中，信息被产生并存储在由编译器在编译报表。在报表，填充时间此信息用于构建net.sf.jasperreports.engine.fill.JRCalculator类的一个实例。

Java源文件生成，并通过对飞基于Java的报表编译器编译。这个生成的类是JRCalculator子类，并通过将其编译产生的字节码存储在JasperReport对象的内部。bytcode被加载在报表填充时间和由此产生的类被实例化，以获得所需的表达式求值计算器对象。

## 条件表达式

定义变量表达式时，Jasper报表不支持if-else语句。相反，可以使用三元运算符{cond}？{语句1}：{语句2}。可以嵌套这个操作符Java表达式里面获得基于多个条件所需的输出。

## 在报表条件表达式的示例

让我们修改现有报告的模板（第报表设计），并增加对country条件表达式。修订后的报表模板（jasper_report_template.jrxml）如下。将其保存到 C:
oolsjasperreports-5.0.1 est 目录:

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC
"//JasperReports//DTD Report Design//EN"
```

```
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
<parameter name="ReportTitle" class="java.lang.String"/>
<parameter name="Author" class="java.lang.String"/>
   <queryString>
    <![CDATA[]]>
   </queryString>
   <field name="country" class="java.lang.String">
      <fieldDescription><![CDATA[country]]></fieldDescription>
   </field>
   <field name="name" class="java.lang.String">
      <fieldDescription><![CDATA[name]]></fieldDescription>
   </field>
   <sortField name="country" order="Descending"/>
   <sortField name="name"/>
   <title>
      <band height="70">
         <line>
            <reportElement x="0" y="0" width="515"
            height="1"/>
         </line>
         <textField isBlankWhenNull="true" bookmarkLevel="1">
            <reportElement x="0" y="10" width="515"
            height="30"/>
            <textElement textAlignment="Center">
            <font size="22"/>
            </textElement>
            <textFieldExpression class="java.lang.String">
            <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>
            <anchorNameExpression><![CDATA["Title"]]>
            </anchorNameExpression>
            </textField>
            <textField isBlankWhenNull="true">
            <reportElement  x="0" y="40" width="515" height="20"/>
            <textElement textAlignment="Center">
                 <font size="10"/>
            </textElement>
            <textFieldExpression class="java.lang.String">
            <![CDATA[$P{Author}]]>
            </textFieldExpression>
            </textField>
      </band>
   </title>
   <columnHeader>
      <band height="23">
         <staticText>
            <reportElement mode="Opaque" x="0" y="3"
            width="535"    height="15"
            backcolor="#70A9A9" />
            <box>
            <bottomPen lineWidth="1.0"
```

```
            lineColor="#CCCCCC" />
         </box>
         <textElement />
         <text><![CDATA[]]>
         </text>
      </staticText>
      <staticText>
         <reportElement x="414" y="3" width="121"
         height="15" />
         <textElement textAlignment="Center"
         verticalAlignment="Middle">
               <font isBold="true" />
         </textElement>
         <text><![CDATA[Country]]></text>
      </staticText>
      <staticText>
         <reportElement x="0" y="3" width="136"
         height="15" />
         <textElement textAlignment="Center"
         verticalAlignment="Middle">
            <font isBold="true" />
         </textElement>
         <text><![CDATA[Name]]></text>
      </staticText>
   </band>
    </columnHeader>
    <detail>
   <band height="16">
      <staticText>
         <reportElement mode="Opaque" x="0" y="0"
         width="535"    height="14"
         backcolor="#E5ECF9" />
         <box>
            <bottomPen lineWidth="0.25"
            lineColor="#CCCCCC" />
         </box>
         <textElement />
         <text><![CDATA[]]>
         </text>
      </staticText>
      <textField>
         <reportElement x="414" y="0" width="121"
         height="15" />
         <textElement textAlignment="Center"
         verticalAlignment="Middle">
            <font size="9" />
         </textElement>
         <textFieldExpression class="java.lang.String"> **<![CDA
         </textFieldExpression>
      </textField>
      <textField>
         <reportElement x="0" y="0" width="136"
         height="15" />
```

```
            <textElement textAlignment="Center"
            verticalAlignment="Middle" />
            <textFieldExpression class="java.lang.String">
            <![CDATA[$F{name}]]>
            </textFieldExpression>
         </textField>
      </band>
   </detail>
</jasperReport>
```

Java代码填充报表如下。该文件的内容 **C: oolsjasperreports-5.0.1 estsrccomyiibaiJasperReportFill.java** 如下所述。

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
      "C://tools/jasperreports-5.0.1/test/jasper_report_template.ja

      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

      try {
         JasperFillManager.fillReportToFile(
         sourceFileName, parameters, beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

POJO文件的内容 **C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBean.java** 情况如下：

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

将增加country字段在Java bean列表为空的新纪录。该文件的内容C:
oolsjasperreports-5.0.1 estsrccomyiibaiDataBeanList.java 情况如下：

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>(),

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California")); **dataBe
      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

## 报表生成

我们将编译和执行使用我们常规Ant构建过程上面的文件。build.xml文件中的内容（根据目录保存：C: oolsjasperreports-5.0.1 est）情况如下。导入文件 - baseBuild.xml，并应放置在同一目录build.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir="
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>
</project>
```

接下来，让我们打开命令行窗口并转到build.xml文件放置的目录。最后执行的命令 **ant -Dmain-class=com.yiibai.JasperReportFill** （viewFullReport是默认的目标），如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

clean-sample:
   [delete] Deleting directory C:    oolsjasperreports-5.0.1    est
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re

compile:
    [mkdir] Created dir: C:    oolsjasperreports-5.0.1    estclasse
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:28:
    warning: 'includeantruntime' was not set, defaulting to build.s
    set to false for repeatable builds
    [javac] Compiling 3 source files to C:    oolsjasperreports-5.0

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly.
      [jrc] log4j:WARN See
      http://logging.apache.org/log4j/1.2/faq.htmll#noconfig for mo
      [jrc] File : C:    oolsjasperreports-5.0.1    estjasper_repor

run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 5 minutes 5 seconds

C:    oolsjasperreports-5.0.1    est>
```

正如上文编译的结果，JasperViewer窗口打开如下面的屏幕：




在这里，可以看到，因为没有通过该字段country任何数据的最后一条记录，“NO COUNTRY”正在打印。

# JasperReports报表变量 - JasperReports教程

报表变量是建立在报表表达式之上的特殊对象。报表变量简化以下任务：

- 报表，其中大量使用在整个报告模板表达式。这些表达式可以通过使用报表变量只能声明一次。
- 计数，求和，平均，最低，最高，方差等：报表变量可以基于像对应的表达式的值执行各种计算

如果变量是在报表设计定义，那么这些可以通过在表达式中的新变量引用。因此，在该变量是在报表设计中声明的顺序是非常重要的。

## 变量声明

变量声明如下：

```
<variable name="CityNumber" class="java.lang.Integer" incrementType
   incrementGroup="CityGroup" calculation="Count">
   <variableExpression>
       <![CDATA[Boolean.TRUE]]>
   </variableExpression>
</variable>
```

如上面所看到的，<variable>元素包含属性的数量。这些属性总结如下：

### name属性

类似的参数和字段，</variable>元素的name属性是强制性的。它允许引用由它的声明的名称在报表表达式中的变量。

### class属性

class属性也是强制性的，它指定了变量值的类名。它的默认值是java.lang.String。这是可以改变的，在classpath可用的任何类无论是在报表编译时间和报告充填时间。无论报告值的类型，引擎采用该$V {}标记是用来报告表达，从而不必要手动投射。

## 计算方法

这个属性决定的变量填充报表时要执行什么计算。下面的小节描述所有的<variable>元素的计算属性的可能值。

- Average: 变量的值是变量表达式的每一个非空值的平均值。有效期仅为数值变量。
- Count: 该变量的值是变量表达式非空实例的数量。
- First: 变量的值是变量表达式的第一个实例的值。随后的值将被忽略。
- Highest: 变量的值是变量表达式的最高值。
- Lowest: 该变量的值是变量表达式在报表中的最低值。
- Nothing: 不进行计算的变量。
- StandardDeviation: 该变量的值是所有非空值匹配报表表达式的标准偏差。有效期仅为数值变量。
- Sum: 该变量的值是由报表表达式返回所有非空值的总和。
- System: 该变量的值是一个自定义计算（计算该变量自己的值，使用JasperReports小脚本功能）
- Variance: 该变量的值是一个报表变量的表达式求值返回的所有非空值的方差。

## 增量FACTORYCLASS

此属性确定填充报表上的当前记录时，用于计算变量的值的类。默认值是任何类实现net.sf.jasperreports.engine.fill.JRIncrementerFactory。工厂类将被用于由发动机来实例化对象的增量在运行时根据该变量中设置的计算属性。

## INCREMENTTYPE

这个决定何时重新计算变量的值。此属性使用的值，如下：

- Column: 该变量的值重新计算各列的结尾
- Group: 当指定incrementGroup改变该组的变量值重新计算。
- None: 该变量的值重新计算每个记录。
- Page: 该变量的值被重新计算在每一页的末尾。
- Report: 该变量的值被重新计算一次，在报表的末尾。

## INCREMENTGROUP

这决定了该变量的值被重新计算，当incrementType是组的名称。这需要在JRXML报表模板中声明的任何组的名称。

## RESETTYPE

这决定了当一个变量的值被复位。此属性使用的值，如下：

- Column: 该变量的值复位在每一列的开头。
- Group: 该变量的值是当指定incrementGroup修改组复位。
- None: 该变量的值不会被重置。
- Page: 该变量的值复位在每一页的开头。
- Report: 该变量的值复位只有一次，在报表的开头。

### RESETGROUP

这决定了该变量的值复位，当resetType是组的名称。该属性的值是在JRXML报表模板中声明的任何组的名称。

## 内置报表变量

有一些内置的系统变量，准备在表达式中使用，如下所示：

| Variable Name | 描述 |
| --- | --- |
| PAGE_NUMBER | 这个变量的值是它的当前页码。它可以被用来同时显示当前页面的数量和使用JasperReports文本字段的元素，evaluationTime属性的一种特殊的功能的总页数。 |
| COLUMN_NUMBER | 这个变量包含了当前的列号 |
| REPORT_COUNT | 此报表变量包含的处理记录的总数。 |
| PAGE_COUNT | 这个变量包含了生成当前页面时所处理的记录数。 |
| COLUMN_COUNT | 这个变量包含了生成当前列时所处理的记录数。 |
| GroupName_COUNT | 这个变量的名称是从它所对应的组，后缀为_COUNT序列的名称派生的。这个变量包含的记录在当前组的数量。 |

## 例子

让一个变量（countNumber）加入到现有的报表模板（第报表设计）。我们将前缀数到每个记录。修订后的报告模板（jasper_report_template.jrxml）如下。将其保存到C: oolsjasperreports-5.0.1 est 目录:

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC
"//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
```

```
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" pageWidth="595"
pageHeight="842" columnWidth="515"
leftMargin="40" rightMargin="40" topMargin="50" bottomMargin="50">
<parameter name="ReportTitle" class="java.lang.String"/>
<parameter name="Author" class="java.lang.String"/>

   <queryString>
    <![CDATA[]]>
   </queryString>

   <field name="country" class="java.lang.String">
      <fieldDescription>
         <![CDATA[country]]>
      </fieldDescription>
   </field>

   <field name="name" class="java.lang.String">
      <fieldDescription>
         <![CDATA[name]]>
      </fieldDescription>
   </field> **<variable name="countNumber" class="java.lang.Integer
      <variableExpression>
         <![CDATA[Boolean.TRUE]]>
      </variableExpression>
   </variable>**
   <title>
      <band height="70">
         <line>
            <reportElement x="0" y="0" width="515"
            height="1"/>
         </line>
         <textField isBlankWhenNull="true" bookmarkLevel="1">
            <reportElement x="0" y="10" width="515"
            height="30"/>
            <textElement textAlignment="Center">
               <font size="22"/>
            </textElement>
            <textFieldExpression class="java.lang.String">
               <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>
            <anchorNameExpression>
               <![CDATA["Title"]]>
            </anchorNameExpression>
         </textField>
         <textField isBlankWhenNull="true">
            <reportElement  x="0" y="40" width="515" height="20"/>
            <textElement textAlignment="Center">
                 <font size="10"/>
```

```
            </textElement>
            <textFieldExpression class="java.lang.String">
               <![CDATA[$P{Author}]]>
            </textFieldExpression>
         </textField>
      </band>
   </title>

   <columnHeader>
      <band height="23">
         <staticText>
            <reportElement mode="Opaque" x="0" y="3"
            width="535"    height="15"
            backcolor="#70A9A9" />
            <box>
            <bottomPen lineWidth="1.0"
            lineColor="#CCCCCC" />
            </box>
            <textElement />
            <text><![CDATA[]]>
            </text>
         </staticText>
         <staticText>
            <reportElement x="414" y="3" width="121"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
                   <font isBold="true" />
            </textElement>
            <text><![CDATA[Country]]></text>
         </staticText>
         <staticText>
            <reportElement x="0" y="3" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font isBold="true" />
            </textElement>
            <text><![CDATA[Name]]></text>
         </staticText>
      </band>
   </columnHeader>

   <detail>
      <band height="16">
         <staticText>
            <reportElement mode="Opaque" x="0" y="0"
            width="535"    height="14"
            backcolor="#E5ECF9" />
            <box>
               <bottomPen lineWidth="0.25"
               lineColor="#CCCCCC" />
            </box>
```

```
            <textElement />
            <text><![CDATA[]]>
            </text>
         </staticText>
         <textField>
            <reportElement x="414" y="0" width="121"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font size="9" />
            </textElement>
            <textFieldExpression class="java.lang.String">
            <![CDATA[$F{country}]]>
            </textFieldExpression>
         </textField>
         <textField>
            <reportElement x="0" y="0" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle" />
            <textFieldExpression class="java.lang.String"> **<![CD/
            </textFieldExpression>
         </textField>
      </band>
   </detail>
</jasperReport>
```

在java代码报表充填保持不变。文件**C: oolsjasperreports-5.0.1 estsrccomyiibaiJasperReportFill.java**内容如下：

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
      "C://tools/jasperreports-5.0.1/test/jasper_report_template.ja

      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

      try {
         JasperFillManager.fillReportToFile(
         sourceFileName, parameters, beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

POJO文件的内容 C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBean.java的内容如下：

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

该文件 **C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBeanList.java** 的内容如下：

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

# 报表生成

我们将编译和执行使用我们常规Ant构建过程上面的文件。 build.xml文件中的内容（根据目录保存在C: oolsjasperreports-5.0.1 est）如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir='
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>
</project>
```

接下来，打开命令行窗口并转到build.xml文件放置的目录。最后执行的命令 **ant -Dmain-class=com.yiibai.JasperReportFill** （viewFullReport默认的目标）， 如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

clean-sample:
   [delete] Deleting directory C:    oolsjasperreports-5.0.1    est
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re

compile:
    [mkdir] Created dir: C:    oolsjasperreports-5.0.1    estclasse
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:28: wa
    'includeantruntime' was not set, defaulting to build.sysclasspa
    set to false for repeatable builds
    [javac] Compiling 7 source files to C:    oolsjasperreports-5.0

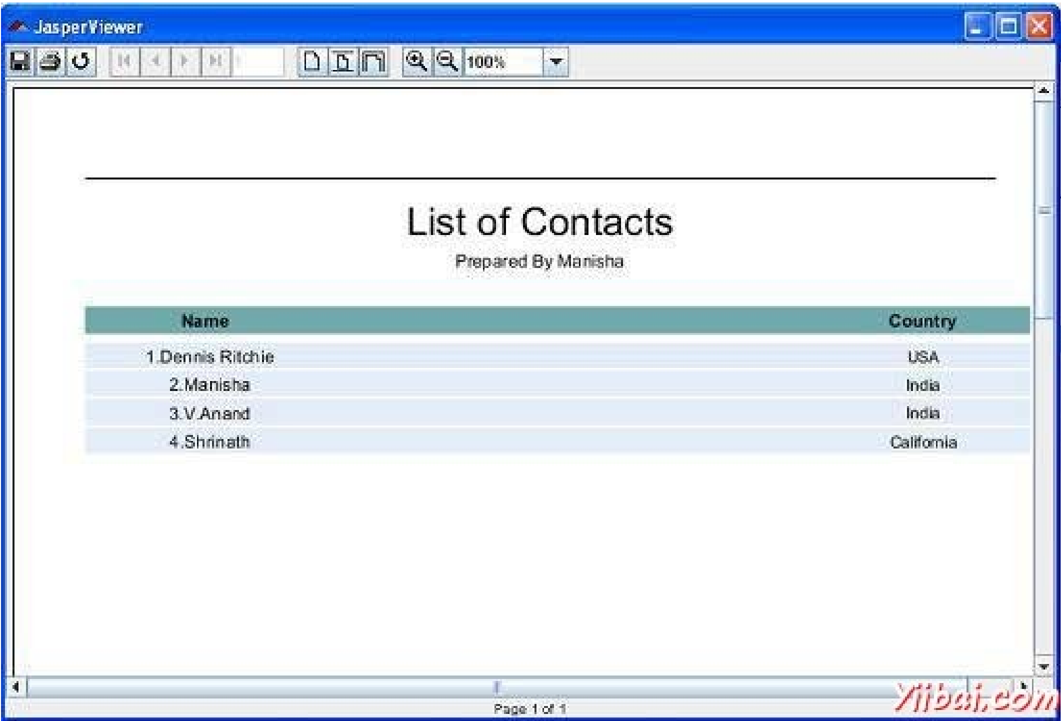
compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly.
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq.
      for more info.
      [jrc] File : C:    oolsjasperreports-5.0.1    estjasper_repor

run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 18 seconds
```

正如上文编译的结果，一个JasperViewer窗口打开如下面的屏幕：




在这里，我们看到，计数前缀为每个记录。

# JasperReports报表区段 - JasperReports教程

我们将在本章开始，一个简单的报表模板的结构看。依样画葫芦JasperReports的结构报表模板归类到多个区段。部分是有规定的高度，并且可以包含像直线，矩形，图像或文本字段对象报表的部分。

通过提供的报表数据源的虚拟记录的报表引擎遍历，在报表填充的时候。根据每个部分的定义的行为，引擎则呈现每个报表节在适当的时候。举例来说，细节部分的数据源中呈现为每个记录。当页中断，页眉和页面页脚节在需要时提供。

在JasperReports术语，报表区段也被称为报表带区。部分是由一个或多个频段。这些部分在报告生成时间反复填充并写在文件最后。

## 主要章节

在JasperReports报表模板主要有以下几个部分：

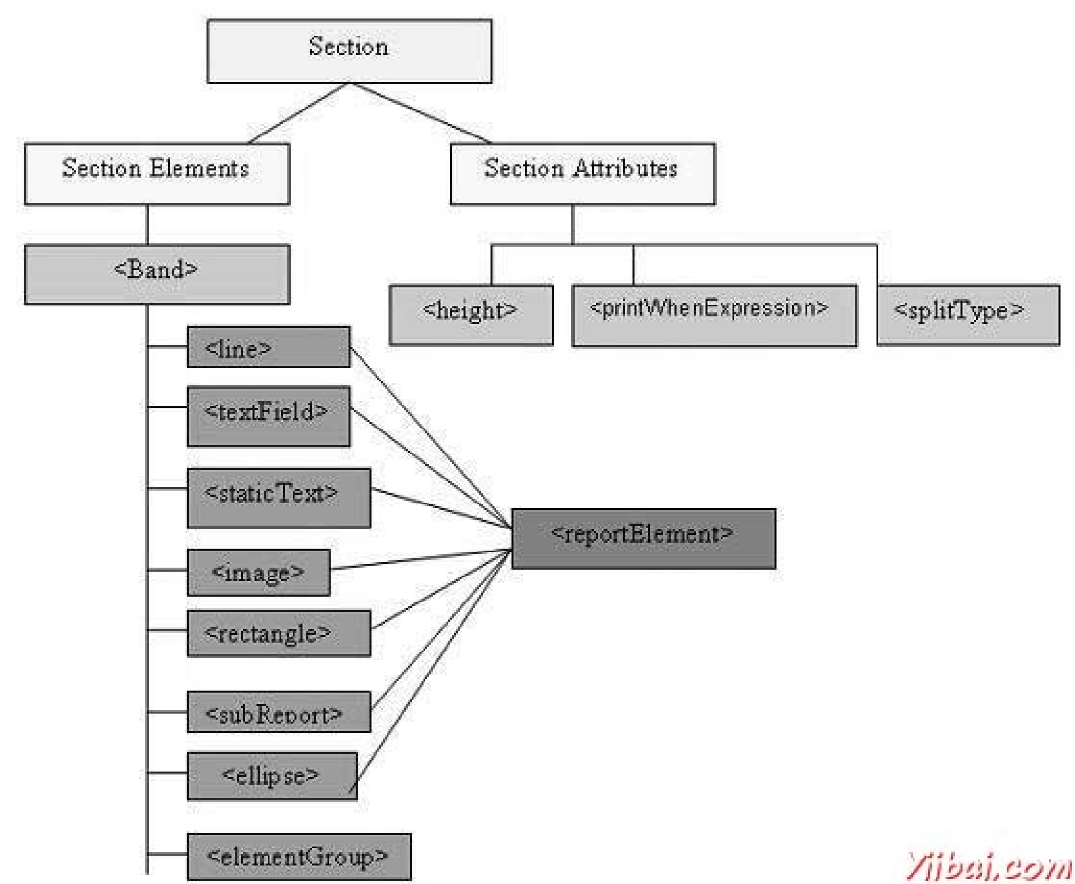
```
<title></title>
<pageheader></pageheader>
<columnheader></columnheader>
<groupheader></groupheader>
<detail></detail>
<groupfooter></groupfooter>
<columnfooter></columnfooter>
<pagefooter></pagefooter>
<lastpagefooter></lastpagefooter>
<summary></summary>
<nodata></nodata<>
<background></background>
```

下表总结了每个部分：

| Section | 描述 |
| --- | --- |
| Title | 本节中只出现一次在报表的开头。 |
| Page Header | 这部分出现在每一页的开头生成的文档中。 |
| Column Header | 这部分显示在生成的文档中的每一列的开头。如果报表只有一个定义列，那么列标题和脚注部分被忽略。 |
| Group Header | 这部分是由一个报告组（组章）引入。每次分组表达式改变其值，组页眉部分上面印的细节部分。在情况下，如果超过一个组被定义，组页眉打印在组定义的顺序。 |
| Detail | 这部分是重复的报表的数据源提供的数据的每一行。细节部分可以由多个频段。 |
| Group Footer | 这部分是由一个报告组（组章）引入。该组页脚节印下的分组表达式的值更改前的细节部分。组页脚始终打印在数据源的数据的最后一行。在情况下，如果超过一个组被定义，组页脚打印在组定义的顺序相反。 |
| Column Footer | 本节将出现在每一列的底部。如果报告的列数为1，则列标题和脚注部分被忽略。 |
| Page Footer | 本节出现在每个页面的底部。 |
| Last Page Footer | 这部分取代了报表的最后一页上的常规页页脚。在情况下，摘要部分也存在，那么这可能不是该文件的最后一页。这部分有时是有用的，当汇总信息具有在最后一页的底部显示。 |
| Summary | 本节中只出现一次在报告的末尾。 |
| No Data | 这部分被打印时，当无资料打印报表属性设置为无数据段。如果<noData>部在报告模板中定义，并且如果数据源是空的，那么<noData>部分将是唯一一个在填充时考虑，其含量将产生报表输出。 |
| Background | 背景部分会显示每一页上，并不能溢出到下一个页面。放在这一部分的元素在页面初始化的时候求值，并显示在背景中。所有其他的页面对象被显示在背景上对象的顶部。这部分是用于创建页面水印有用。 |

## 部分，元素和属性的关系

下图显示的元素和属性在报表中的部分关系。




## section元素

所有上述报表部分都是可选的。但任何报表模板将至少有一个这样的部分。每一节都包含一个单一的<band>元素作为其唯一的子元素。一个<band>可以包含零个或多个下列子元素：

, , , , , , , or

这些元素都必须包含一个<reportElement>作为其第一个元素（除了elementGroup）。一个<reportElement>决定了数据是如何奠定了该特定元素。与变量和参数，不要求报表内容有一个名字，因为通常不需要获得一个报表模板内的任何单个元素。

下表总结了<reportElement>属性：

| 属性 | 描述 | 有效值 |
| --- | --- | --- |
| x | 指定频带元件的x坐标。 | 一个整数值，表示以像素为单位的元素的x坐标。此属性是必需的。 |
| y | 指定频带元件的y坐标。 | 一个整数值，表示在y以像素为元素的坐标。此属性是必需的。 |

| | | |
|---|---|---|
| width | 指定频带元件的宽度。 | 一个整数值，表示该元素的宽度以像素为单位。此属性是必需的。 |
| height | 指定频带元件的高度。 | 一个整数值，表示以像素为元素的高度。此属性是必需的。 |
| key | 带元素的唯一标识符。 | 唯一字符串值。 |
| stretchType | 指定包含带延伸当元素如何延伸 | **NoStretch (default)**: 该元素不会延长。**RelativeToTallestObject**: 该元素将伸展以适应它的组的最高的对象。**RelativeToBand**: 该元素将延伸到适合带的高度。 |
| positionType | 指定当频带延伸元素的位置。 | **Float**: 元素将取决于周围元件的尺寸移动。**FixRelativeToTop (default)**: 该元素将保持一个固定的位置相对于带的顶部。**FixRelativeToBottom**: 该元素将保持一个固定的位置相对于带的底部。 |
| isPrintRepeatedValues | 如果指定的值重复打印。 | **true (default)**: 重复的值将被打印出来。**false**: 重复的值将不被打印出来。 |
| mode | 指定元素的背景模式 | 不透明的，透明的 |
| isRemoveLineWhenBlank | 如果指定了当它是空白，并有在相同的水平空间没有其他元素的元素应被删除。 | true, false |
| isPrintInFirstWholeBand | 如果指定的元素必须打印在整个频段，也就是说，未被报表的页面或列之间分割的波段。 | true, false |

| | | |
|---|---|---|
| isPrintWhenDetailOverFlows | 指定是否当频带溢出到新页或列中的元素将被打印出来。 | true, false |
| printWhenGroupChanges | 指定在指定的组改变元素将被打印。 | string值 |
| forecolor | 指定元素的前景色。 | 无论前面加上#字符，或以下预定义值中的一个十六进制RGB值：*black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, yellow, white.* |
| backcolor | 指定元素的背景颜色。 | 同样作为有效值的前景色 |

# 段属性

以下是报表的部分属性：

## 高度

该部分的高度指定高度，该特定部分的像素，是非常重要的在整体报表设计。

## 打印当表达式

布尔表达式，确定该部分是否应打印或不打印。

## SPLIT ALLOWED

一个标志，指示该部分是否允许分裂时，它不适合在当前页面上。如果为true，该部分将被转移到下一个页面。注意，如果一节不适合下页上，那么将会考虑该标志的值的拆分。 splitType可以利用以下值：

- splitType="Stretch" : 拆分拉伸内容。如果该部分在当前页上延伸（如果可用空间小于宣高度），这是添加到原始高度的区域是否允许分割到下页
- splitType="Prevent" : 避免在第一次尝试分割。如果部分不适合在下一页中，分割通常发生，预防频带分割是有效的只有在第一次分割尝试。
- splitType="Immediate" : 立即绘制。该频段允许的任何地方，除了分割高于其最顶端的元素。

# 例子

为演示开始每个部分让我们编写报表模板（jasper_report_template.jrxml）。将此文件保存到C: oolsjasperreports-5.0.1 est目录。在这个文件中，我们将显示在每个部分的文本（上面所讨论的）。该文件的内容如下所述：

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" pageWidth="300" pageHeight="300" colu
leftMargin="0" rightMargin="0" topMargin="0" bottomMargin="0" >

   <title>
      <band height="50">
         <textField>
            <reportElement x="100" y="16"
            width="100" height="20"/>
            <textElement/>
            <textFieldExpression>
               <![CDATA["Title"]]>
            </textFieldExpression>
         </textField>
      </band>
   </title>
   <pageHeader>
      <band height="40">
         <textField>
            <reportElement  mode="Opaque" x="100" y="10"
            width="90" height="20"/>
            <textElement>
               <font isBold="true"/>
            </textElement>
            <textFieldExpression>
               <![CDATA["Page Header"]]>
            </textFieldExpression>
         </textField>
      </band>
   </pageHeader>
   <columnHeader>
      <band height="40">
            <textField>
            <reportElement  x="100" y="10" width="90"
            height="20"/>
            <textElement>
               <font isItalic="true"/>
            </textElement>
            <textFieldExpression>
               <![CDATA["Column Header"]]>
            </textFieldExpression>
```

```
            </textField>
         </band>
      </columnHeader>
      <detail>
         <band height="40">
            <textField>
               <reportElement mode="Opaque" x="100" y="10" width="90"
               height="20" backcolor="#99CCFF"/>
               <textElement/>
               <textFieldExpression>
                  <![CDATA["Report Details"]]>
               </textFieldExpression>
            </textField>
         </band>
      </detail>
      <columnFooter>
         <band height="40">
            <textField>
               <reportElement  x="100" y="10" width="90"
               height="20"/>
               <textElement/>
               <textFieldExpression>
                  <![CDATA["Column Footer"]]>
               </textFieldExpression>
            </textField>
         </band>
      </columnFooter>
      <pageFooter>
         <band height="40">
            <textField>
               <reportElement  x="100" y="10"
               width="90" height="20"/>
               <textElement/>
               <textFieldExpression>
                  <![CDATA["Page Footer"]]>
               </textFieldExpression>
            </textField>
         </band>
      </pageFooter>
      <lastPageFooter>
         <band height="40">
            <textField>
               <reportElement  x="100" y="10" width="90"
               height="20"/>
               <textElement/>
               <textFieldExpression>
                  <![CDATA["Last Page Footer"]]>
               </textFieldExpression>
               </textField>
         </band>
      </lastPageFooter>
      <summary>
         <band height="40">
```

```
            <textField>
               <reportElement  x="100" y="10" width="90"
               height="20"/>
               <textElement/>
               <textFieldExpression>
                  <![CDATA["Summary"]]>
               </textFieldExpression>
            </textField>
         </band>
      </summary>
   </jasperReport>
```

java代码填写并生成报告如下。保存这个文件JasperReportFill.java 到 C: oolsjasperreports-5.0.1 estsrccomyiibai 目录.

```
package com.yiibai;

import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;

public class JasperReportFill {
   public static void main(String[] args) {
      String sourceFileName =
            "C://tools/jasperreports-5.0.1/test/" +
            "jasper_report_template.jasper";

      try {
         JasperFillManager.fillReportToFile(sourceFileName, null,
            new JREmptyDataSource());
      } catch (JRException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }
}
```

在这里，我们填充报表时，以模拟一个数据源，在这一个记录使用JREmptyDataSource的实例，但与所有在这个单一记录为null的字段。

## 报表生成

将编译和执行使用常规Ant构建过程上面的文件build.xml文件中的内容（根据目录保存：C: oolsjasperreports-5.0.1 est）情况如下。(saved under directory C: oolsjasperreports-5.0.1 est) 如下.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir='
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>
</project>
```

接下来，让我们打开命令行窗口并转到build.xml文件放置的目录。最后执行的命令 **ant -Dmain-class=com.yiibai.JasperReportFill**（viewFullReport是默认的目标），如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

clean-sample:
   [delete] Deleting directory C:    oolsjasperreports-5.0.1    est
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re

compile:
    [mkdir] Created dir: C:    oolsjasperreports-5.0.1    estclasse
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:28:
    warning: 'includeantruntime' was not set, defau
    [javac] Compiling 1 source file to C:    oolsjasperreports-5.0

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFac
      [jrc] log4j:WARN Please initialize the log4j system properly
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq
      [jrc] File : C:    oolsjasperreports-5.0.1    estjasper_repor

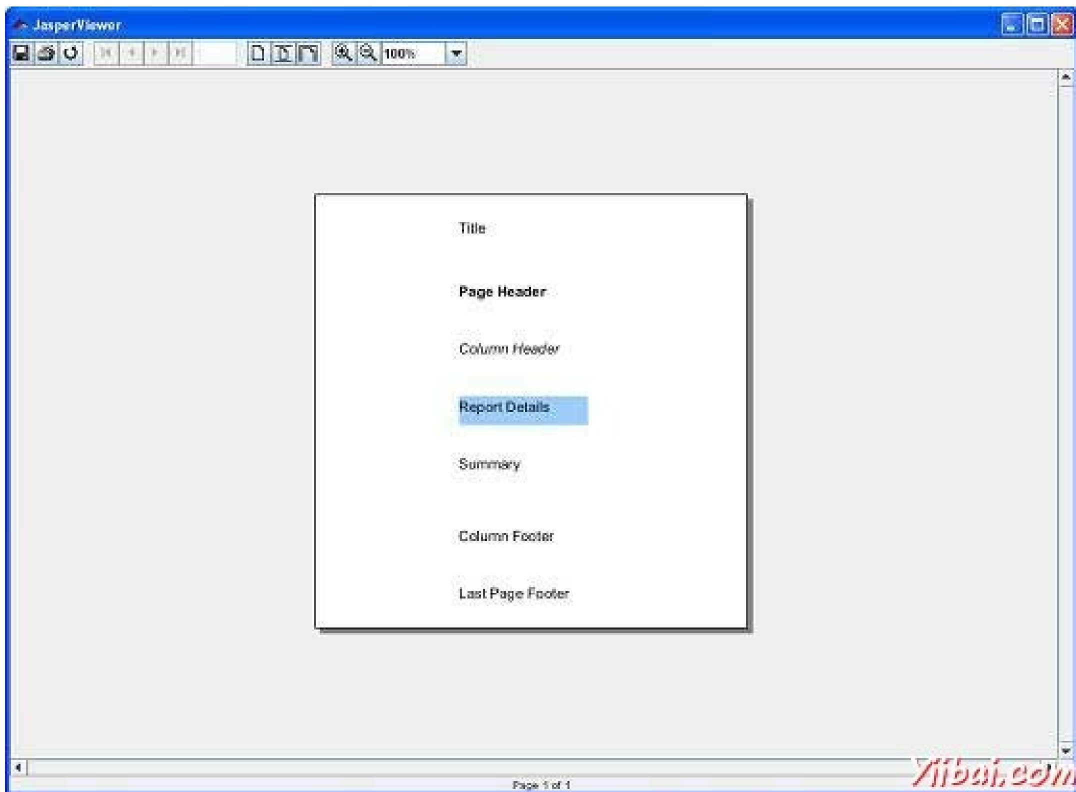
run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnviro
     [java] log4j:WARN Please initialize the log4j system properly

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnviro
     [java] log4j:WARN Please initialize the log4j system properly

BUILD SUCCESSFUL
Total time: 18 minutes 22 seconds
```



正如上文编译的结果，JasperViewer窗口打开如下面的屏幕：




在这里，我们可以在每个文本被打印的部分看到。但应当注意的是，作为JRXML包含<lastPageFooter>元件，它将被显示在报表，而不是所显示的<pageFooter>元素的最后一页。如果有多个列的<columnHeader>和<columnFooter>元素将只显示在报告中。

# JasperReports报表组 - JasperReports教程

组在JasperReports的协助组织对报告的数据以逻辑方式。报告组代表连续记录的数据源中有一些共同点，比如某个报表字段的值的序列。报告组由<group>元素定义。一个报表可以有任意数量的组。一旦声明，群体可以在整个报告中提到的。

报告组有三个要素：

- Group expression: 这表示必须改变，以启动一个新的数据组中的数据。
- Group header section: 帮助位置标签在分组数据的开始。
- Group footer section: : 帮助位置标签在分组数据的末尾。

在截至于若该组表达变化，一组发生断裂和相应的<groupFooter>和<groupHeader>部分的值插入到生成的文档中报告充填时数据源的迭代。

> 报表组的机制不执行由数据源所提供的资料的任何排序。数据分组按预期工作，只有当数据源中的记录按照报告中使用的组表达式已经下令。

## 属性组

<group>元素包含属性，使我们能够控制分组的数据是如何布局。属性概括于下表：

| 属性 | 描述 |
| --- | --- |
| name | 这是强制性的。它通过名称引用该组中的报表表达式。它遵循相同的命名约定我们，我们提到的报告参数，字段和报表变量。它可以在其他JRXML属性被用于当想引用一个特定的报告组。 |
| isStartNewColumn | 当设置为true时，每个数据组将开始一个新的列。默认值是false |
| isStartNewPage | 当设置为true时，每个数据组将开始一个新的页面上。默认值是false |
| isResetPageNumber | 当设置为true，该报告页码将每一个新组开始时被重置。默认值是false |
| isReprintHeaderOnEachPage | 当设置为true时，组头会被重印每一页上。默认值是false |
| minHeightToStartNewPage | 定义在列的底部，以便将组头当前列所需要的垂直空间最小量。被指定在报告单位的数量。 |
| footerPosition | 呈现在页面上的组页脚的位置，以及其有关的报告的部分它后面的行为。它的值可以是: *Normal*, *StackAtBottom*, *ForceAtBottom*, *CollateAtBottom*.默认值是 *Normal* |
| keepTogether | 当设置为true，将阻止该集团从分割它第一次突破的尝试 |

# 例子

让我们添加一个组（CountryGroup）现有的报告模板（章报表设计）。每个国家的次数进行计数，计数显示为组页脚。在组头中每个记录的计数前缀。修订后的报告模板（jasper_report_template.jrxml）如下。将其保存到 C: oolsjasperreports-5.0.1 est 目录：

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC
"//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperrep
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperrepo
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" pageWidth="595"
pageHeight="842" columnWidth="515"
leftMargin="40" rightMargin="40" topMargin="50" bottomMargin="50">
```

```
<parameter name="ReportTitle" class="java.lang.String"/>
<parameter name="Author" class="java.lang.String"/>

   <queryString>
    <![CDATA[]]>
   </queryString>

   <field name="country" class="java.lang.String">
      <fieldDescription><![CDATA[country]]></fieldDescription>
   </field>

   <field name="name" class="java.lang.String">
      <fieldDescription><![CDATA[name]]></fieldDescription>
   </field>
   <sortField name="country" order="Descending"/>
   <sortField name="name"/>
   <variable name="CountryNumber" class="java.lang.Integer"
      incrementType="Group" incrementGroup="CountryGroup"
      calculation="Count">
      <variableExpression><![CDATA[Boolean.TRUE]]></variableExpress
   </variable> **<group name="CountryGroup" minHeightToStartNewPage
      <groupExpression><![CDATA[$F{country}]]></groupExpression>
      <groupHeader>
         <band height="20">
            <textField evaluationTime="Group" evaluationGroup="Coun
               bookmarkLevel="1">
               <reportElement mode="Opaque" x="0" y="5" width="515"
                  height="15" backcolor="#C0C0C0"/>
               <box leftPadding="10">
                  <bottomPen lineWidth="1.0"/>
               </box>
               <textElement/>
               <textFieldExpression class="java.lang.String">
                  <![CDATA["  " + String.valueOf($V{CountryNumber})
                  + String.valueOf($F{country})]]>
               </textFieldExpression>
               <anchorNameExpression>
                  <![CDATA[String.valueOf($F{country})]]>
               </anchorNameExpression>
            </textField>
         </band>
      </groupHeader>
      <groupFooter>
         <band height="20">
            <staticText>
               <reportElement x="400" y="1" width="60" height="15",
               <textElement textAlignment="Right"/>
               <text><![CDATA[Count :]]></text>
            </staticText>
            <textField>
               <reportElement x="460" y="1" width="30" height="15",
               <textElement textAlignment="Right"/>
               <textFieldExpression class="java.lang.Integer">
```

```
                    <![CDATA[$V{CountryGroup_COUNT}]]>
                 </textFieldExpression>
              </textField>
           </band>
        </groupFooter>
     </group>**
     <title>
        <band height="70">
           <line>
              <reportElement x="0" y="0" width="515"
              height="1"/>
           </line>
           <textField isBlankWhenNull="true" bookmarkLevel="1">
              <reportElement x="0" y="10" width="515"
              height="30"/>
              <textElement textAlignment="Center">
              <font size="22"/>
              </textElement>
              <textFieldExpression class="java.lang.String">
              <![CDATA[$P{ReportTitle}]]>
              </textFieldExpression>
              <anchorNameExpression><![CDATA["Title"]]>
              </anchorNameExpression>
              </textField>
              <textField isBlankWhenNull="true">
              <reportElement  x="0" y="40" width="515" height="20"/>
              <textElement textAlignment="Center">
                   <font size="10"/>
              </textElement>
              <textFieldExpression class="java.lang.String">
              <![CDATA[$P{Author}]]>
              </textFieldExpression>
              </textField>
        </band>
     </title>

     <columnHeader>
        <band height="23">
           <staticText>
              <reportElement mode="Opaque" x="0" y="3"
              width="535"    height="15"
              backcolor="#70A9A9" />
              <box>
              <bottomPen lineWidth="1.0"
              lineColor="#CCCCCC" />
              </box>
              <textElement />
              <text><![CDATA[]]>
              </text>
           </staticText>
           <staticText>
              <reportElement x="414" y="3" width="121"
              height="15" />
```

```
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
                   <font isBold="true" />
            </textElement>
            <text><![CDATA[Country]]></text>
         </staticText>
         <staticText>
            <reportElement x="0" y="3" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font isBold="true" />
            </textElement>
            <text><![CDATA[Name]]></text>
         </staticText>
      </band>
   </columnHeader>

   <detail>
      <band height="16">
         <staticText>
            <reportElement mode="Opaque" x="0" y="0"
            width="535"    height="14"
            backcolor="#E5ECF9" />
            <box>
               <bottomPen lineWidth="0.25"
               lineColor="#CCCCCC" />
            </box>
            <textElement />
            <text><![CDATA[]]>
            </text>
         </staticText>
         <textField>
            <reportElement x="414" y="0" width="121"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font size="9" />
            </textElement>
            <textFieldExpression class="java.lang.String">
            <![CDATA[$F{country}]]>
            </textFieldExpression>
         </textField>
         <textField>
            <reportElement x="0" y="0" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle" />
            <textFieldExpression class="java.lang.String">
            <![CDATA[$F{name}]]>
            </textFieldExpression>
         </textField>
      </band>
```

```
   </detail>
</jasperReport>
```

在java代码报表填充保持不变。该文件的内容 **C: oolsjasperreports-5.0.1 estsrccomyiibaiJasperReportFill.java** 如下：

```
package com.yiibai;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
      "C://tools/jasperreports-5.0.1/test/jasper_report_template.ja

      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList()

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

      try {
         JasperFillManager.fillReportToFile(
         sourceFileName, parameters, beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

该POJO文件的内容 C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBean.java 如下：

```
package com.yiibai;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

该文件的内容C: oolsjasperreports-5.0.1 estsrccomyiibaiDataBeanList.java 如下：

```
package com.yiibai;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

# 报表生成

我们将编译和执行使用我们常规Ant构建过程上面的文件.build.xml文件中的内容（根据目录保存 C: oolsjasperreports-5.0.1 est）如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir='
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath refid="classpath" />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath refid="classpath" />
      </jrc>
   </target>
</project>
```

接下来，让我们打开命令行窗口并转到build.xml文件放置的目录。最后执行的命令 ant -Dmain-class=com.yiibai.JasperReportFill （viewFullReport是默认的目标），如下所示：

```
C:    oolsjasperreports-5.0.1    est>ant -Dmain-class=com.yiibai.Ja
Buildfile: C:    oolsjasperreports-5.0.1    estuild.xml

clean-sample:
   [delete] Deleting directory C:    oolsjasperreports-5.0.1    est
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re
   [delete] Deleting: C:    oolsjasperreports-5.0.1    estjasper_re

compile:
    [mkdir] Created dir: C:    oolsjasperreports-5.0.1    estclasse
    [javac] C:    oolsjasperreports-5.0.1    estaseBuild.xml:28: wa
    'includeantruntime' was not set, defaulting to build.sysclasspa
    set to false for repeatable builds
    [javac] Compiling 7 source files to C:    oolsjasperreports-5.0

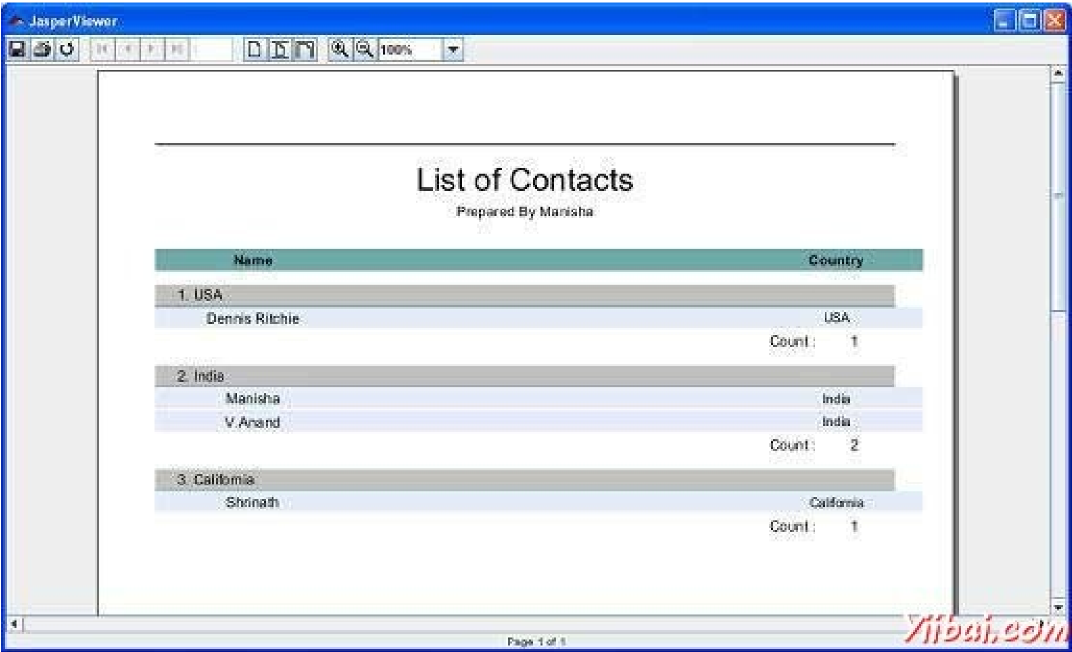
compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly.
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq.
      for more info.
      [jrc] File : C:    oolsjasperreports-5.0.1    estjasper_repor

run:
     [echo] Runnin class : com.yiibai.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 18 seconds
```

正如上文编译的结果，JasperViewer窗口打开如下面的屏幕：




在这里，我们看到，各个国家分组和发生每个国家的计数显示在每个组页脚。

# Java 教程

# Java 基础

# Java 简介

Java是由Sun Microsystems公司于1995年5月推出的Java面向对象程序设计语言和Java平台的总称。由James Gosling和同事们共同研发，并在1995年正式推出。

Java分为三个体系：

- JavaSE（J2SE）（Java2 Platform Standard Edition，java平台标准版）
- JavaEE(J2EE)(Java 2 Platform,Enterprise Edition，java平台企业版)
- JavaME(J2ME)(Java 2 Platform Micro Edition，java平台微型版)。

2005年6月，JavaOne大会召开，SUN公司公开Java SE 6。此时，Java的各种版本已经更名以取消其中的数字"2"：J2EE更名为Java EE, J2SE更名为Java SE，J2ME更名为Java ME。

## 主要特性

- **Java**语言是简单的：

  Java语言的语法与C语言和C++语言很接近，使得大多数程序员很容易学习和使用。另一方面，Java丢弃了C++中很少使用的、很难理解的、令人迷惑的那些特性，如操作符重载、多继承、自动的强制类型转换。特别地，Java语言不使用指针，而是引用。并提供了自动的废料收集，使得程序员不必为内存管理而担忧。

- **Java**语言是面向对象的：

  Java语言提供类、接口和继承等原语，为了简单起见，只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为implements）。Java语言全面支持动态绑定，而C++语言只对虚函数使用动态绑定。总之，Java语言是一个纯的面向对象程序设计语言。

- **Java**语言是分布式的：

  Java语言支持Internet应用的开发，在基本的Java应用编程接口中有一个网络应用编程接口（java net），它提供了用于网络应用编程的类库，包括URL、URLConnection、Socket、ServerSocket等。Java的RMI（远程方法激活）机制也是开发分布式应用的重要手段。

- **Java**语言是健壮的：

  Java的强类型机制、异常处理、垃圾的自动收集等是Java程序健壮性的重要保证。对指针的丢弃是Java的明智选择。Java的安全检查机制使得Java更具健壮性。

- **Java**语言是安全的：

Java通常被用在网络环境中，为此，Java提供了一个安全机制以防恶意代码的攻击。除了Java语言具有的许多安全特性以外，Java对通过网络下载的类具有一个安全防范机制（类ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查，并提供安全管理机制（类SecurityManager）让Java应用设置安全哨兵。

- **Java**语言是体系结构中立的：

  Java程序（后缀为java的文件）在Java平台上被编译为体系结构中立的字节码格式（后缀为class的文件），然后可以在实现这个Java平台的任何系统中运行。这种途径适合于异构的网络环境和软件的分发。

- **Java**语言是可移植的：

  这种可移植性来源于体系结构中立性，另外，Java还严格规定了各个基本数据类型的长度。Java系统本身也具有很强的可移植性，Java编译器是用Java实现的，Java的运行环境是用ANSI C实现的。

- **Java**语言是解释型的：

  如前所述，Java程序在Java平台上被编译为字节码格式，然后可以在实现这个Java平台的任何系统中运行。在运行时，Java平台中的Java解释器对这些字节码进行解释执行，执行过程中需要的类在联接阶段被载入到运行环境中。

- **Java**是高性能的：

  与那些解释型的高级脚本语言相比，Java的确是高性能的。事实上，Java的运行速度随着JIT(Just-In-Time）编译器技术的发展越来越接近于C++。

- **Java**语言是多线程的：

  在Java语言中，线程是一种特殊的对象，它必须由Thread类或其子（孙）类来创建。通常有两种方法来创建线程：其一，使用型构为Thread(Runnable)的构造子将一个实现了Runnable接口的对象包装成一个线程，其二，从Thread类派生出子类并重写run方法，使用该子类创建的对象即为线程。值得注意的是Thread类已经实现了Runnable接口，因此，任何一个线程均有它的run方法，而run方法中包含了线程所要运行的代码。线程的活动由一组方法来控制。Java语言支持多个线程的同时执行，并提供多线程之间的同步机制（关键字为synchronized）。

- **Java**语言是动态的：

  Java语言的设计目标之一是适应于动态变化的环境。Java程序需要的类能够动态地被载入到运行环境，也可以通过网络来载入所需要的类。这也有利于软件的升级。另外，Java中的类有一个运行时刻的表示，能进行运行时刻的类型检查。

## 发展历史

- 1995年5月23日，Java语言诞生
- 1996年1月，第一个JDK-JDK1.0诞生

- 1996年4月，10个最主要的操作系统供应商申明将在其产品中嵌入JAVA技术
- 1996年9月，约8.3万个网页应用了JAVA技术来制作
- 1997年2月18日，JDK1.1发布
- 1997年4月2日，JavaOne会议召开，参与者逾一万人，创当时全球同类会议规模之纪录
- 1997年9月，JavaDeveloperConnection社区成员超过十万
- 1998年2月，JDK1.1被下载超过2,000,000次
- 1998年12月8日，JAVA2企业平台J2EE发布
- 1999年6月，SUN公司发布Java的三个版本：标准版（JavaSE,以前是J2SE）、企业版（JavaEE以前是J2EE）和微型版（JavaME，以前是J2ME）
- 2000年5月8日，JDK1.3发布
- 2000年5月29日，JDK1.4发布
- 2001年6月5日，NOKIA宣布，到2003年将出售1亿部支持Java的手机
- 2001年9月24日，J2EE1.3发布
- 2002年2月26日，J2SE1.4发布，自此Java的计算能力有了大幅提升
- 2004年9月30日18:00PM，J2SE1.5发布，成为Java语言发展史上的又一里程碑。为了表示该版本的重要性，J2SE1.5更名为Java SE 5.0
- 2005年6月，JavaOne大会召开，SUN公司公开Java SE 6。此时，Java的各种版本已经更名，以取消其中的数字"2"：J2EE更名为Java EE，J2SE更名为Java SE，J2ME更名为Java ME
- 2006年12月，SUN公司发布JRE6.0
- 2009年04月20日，甲骨文74亿美元收购Sun。取得java的版权。
- 2010年11月，由于甲骨文对于Java社区的不友善，因此Apache扬言将退出JCP[4]。
- 2011年7月28日，甲骨文发布java7.0的正式版。

## Java开发工具

Java语言尽量保证系统内存在1G以上，其他工具如下所示：

- Linux 系统或者Windows 95/98/2000/XP，WIN 7/8系统
- Java JDK 7
- Notepad编辑器或者其他编辑器。
- IDE：Eclipse

安装好以上的工具后，我们就可以输出Java的第一个程序"Hello World！"

```
public class MyFirstJavaProgram {

    public static void main(String []args) {
       System.out.println("Hello World");
    }
}
```

在下一章节我们将介绍如何配置java开发环境。

# Java开发环境配置

在本章节中我们将为大家介绍如何搭建Java开发环境。

## window系统安装java

### 下载JDK

首先我们需要下载java开发工具包JDK，下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index.html，点击如下下载按钮：

下载后JDK的安装根据提示进行，还有安装JDK的时候也会安装JRE，一并安装就可以了。

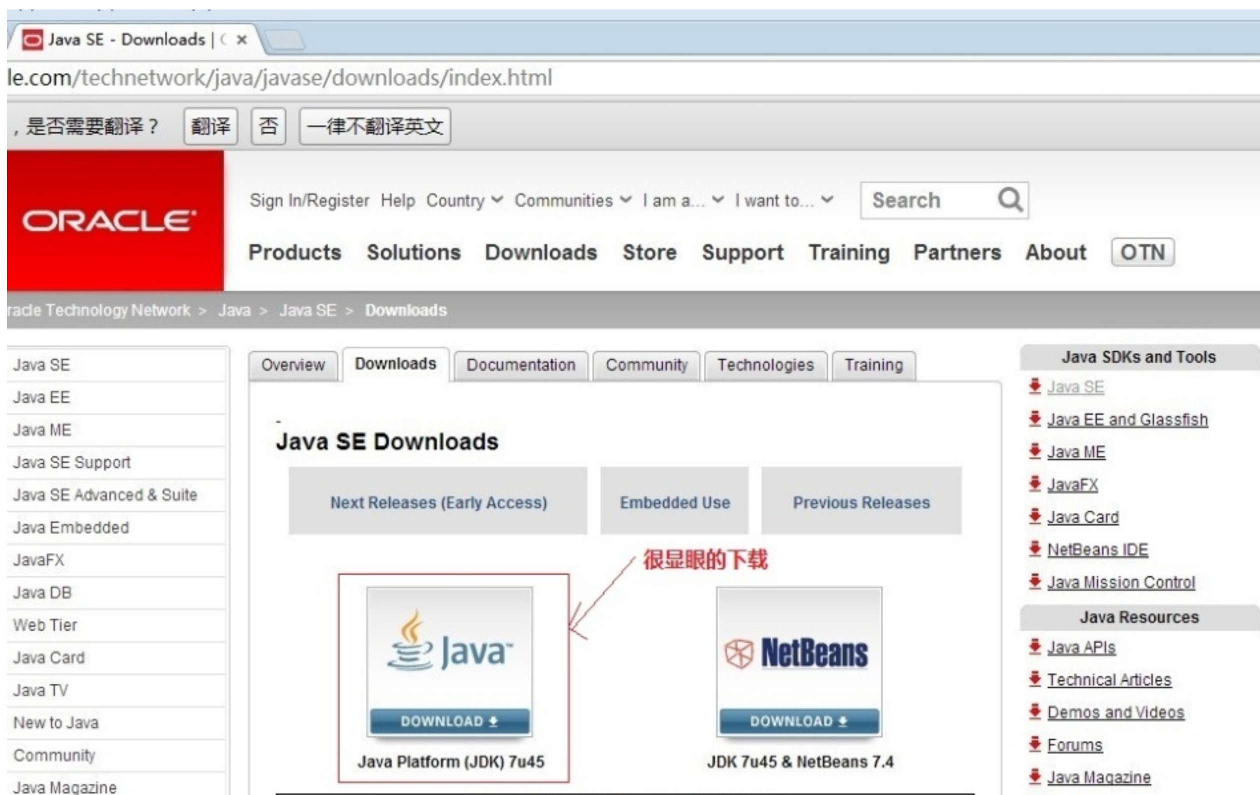
安装JDK，安装过程中可以自定义安装目录等信息，例如我们选择安装目录为 C:\Program Files\Java\jdk1.7.0 。

### 配置环境变量

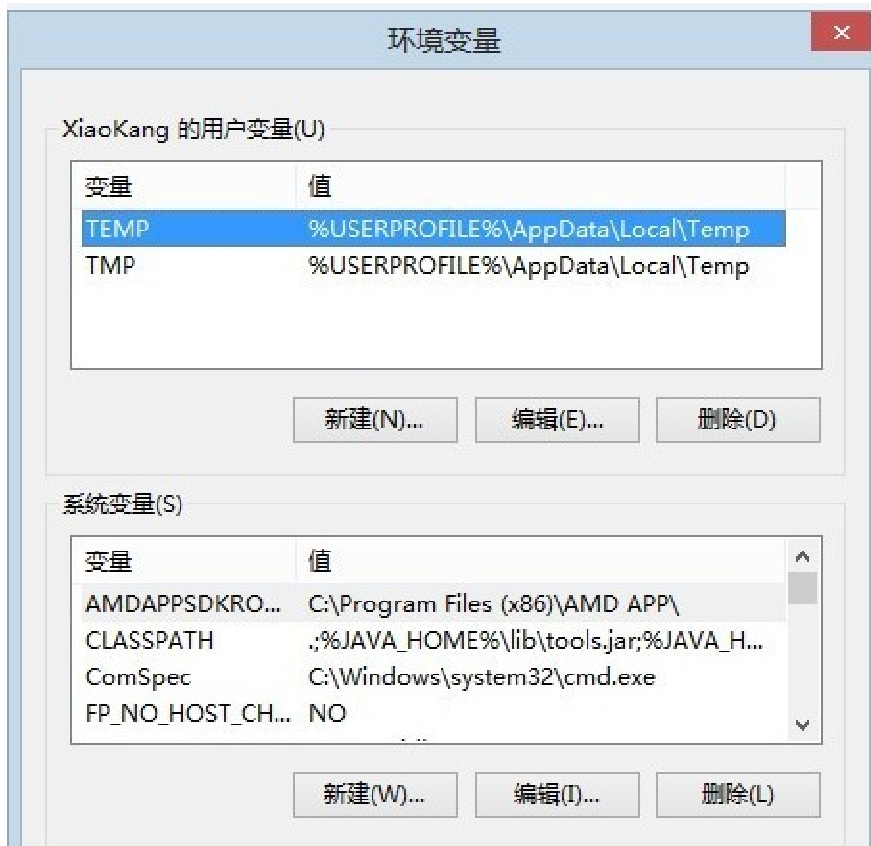
1.安装完成后，右击"我的电脑"，点击"属性"；

2.选择"高级"选项卡，点击"环境变量"；

然后就会出现如下图所示的画面




在"系统变量"中设置3项属性，JAVA_HOME,PATH,CLASSPATH(大小写无所谓),若已存在则点击"编辑"，不存在则点击"新建"。




变量设置

- 变量名：JAVA_HOME

- 变量值：C:\Program Files\Java\jdk1.7.0

//这里是你JDK的安装路径，可以更换

- 变量名：CLASSPATH
- 变量值：.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar; //记得前面有个"."

- 变量名：Path

- 变量值：%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;

这是java的环境配置，配置完成后直接启动eclipse，它会自动完成java环境的配置。

## 测试JDK是否安装成功

1、"开始"->;"运行"，键入"cmd"；

2、键入命令"java -version"，"java"，"javac"几个命令，出现画面，说明环境变量配置成功；

```
C:\Users\XiaoKang>java
用法: java [-options] class [args...]
           (执行类)
   或  java [-options] -jar jarfile [args...]
           (执行 jar 文件)
其中选项包括:
    -d32          使用 32 位数据模型 (如果可用)
    -d64          使用 64 位数据模型 (如果可用)
    -server       选择 "server" VM
    -hotspot      是 "server" VM 的同义词 [已过时]
                  默认 VM 是 server.

    -cp <目录和 zip/jar 文件的类搜索路径>
    -classpath <目录和 zip/jar 文件的类搜索路径>
                  用 ; 分隔的目录, JAR 档案
                  和 ZIP 档案列表, 用于搜索类文件。
    -D<name>=<value>
                  设置系统属性
    -verbose[:class|gc|jni]
                  启用详细输出
```

# Linux，UNIX，Solaris，FreeBSD环境变量设置

环境变量PATH应该设定为指向Java二进制文件安装的位置。如果设置遇到困难，请参考shell文档。

例如，假设你使用bash作为shell，你可以把下面的内容添加到你的 .bashrc文件结尾: export PATH=/path/to/java:$PATH

## 流行JAVA开发工具

正所谓工欲善其事必先利其器，我们在开发java语言过程中同样需要依款不错的开发工具，目前市场上的IDE很多，本文为大家推荐一下几款java开发工具：

- **Notepad++ :** Notepad++ 是在微软视窗环境之下的一个免费的代码编辑器，下载地址： http://notepad-plus-plus.org/
- **Netbeans:**开源免费的java IDE，下载地址： http://www.netbeans.org/index.html
- **Eclipse:**另一个免费开源的java IDE，下载地址： http://www.eclipse.org/

# Java基础语法

一个Java程序可以认为是一系列对象的集合，而这些对象通过调用彼此的方法来协同工作。下面简要介绍下类、对象、方法和实例变量的概念。

- 对象：对象是类的一个实例，有状态和行为。例如，一条狗是一个对象，它的状态有：颜色、名字、品种；行为有：摇尾巴、叫、吃等。
- 类：类是一个模板，它描述一类对象的行为和状态。
- 方法：方法就是行为，一个类可以有很多方法。逻辑运算、数据修改以及所有动作都是在方法中完成的。
- 实例变量：每个对象都有独特的实例变量，对象的状态由这些实例变量的值决定。

## 第一个Java程序

下面看一个简单的Java程序，它将打印字符串 *Hello World*

```
public class MyFirstJavaProgram {
   /* 第一个Java程序.
    * 它将打印字符串 Hello World
    */
    public static void main(String []args) {
       System.out.println("Hello World"); // 打印 Hello World
    }
}
```

下面将逐步介绍如何保存、编译以及运行这个程序：

- 打开Notepad，把上面的代码添加进去；
- 把文件名保存为：MyFirstJavaProgram.java；
- 打开cmd命令窗口，进入目标文件所在的位置，假设是C:\
- 在命令行窗口键入 javac MyFirstJavaProgram.java 按下enter键编译代码。如果代码没有错误，cmd命令提示符会进入下一行。（假设环境变量都设置好了）。
- 再键入java MyFirstJavaProgram 按下Enter键就可以运行程序了

你将会在窗口看到 Hello World

```
C : > javac MyFirstJavaProgram.java
C : > java MyFirstJavaProgram
Hello World
```

## 基本语法

编写Java程序时，应注意以下几点：

- 大小写敏感：Java是大小写敏感的，这就意味着标识符Hello与hello是不同的。
- 类名：对于所有的类来说，类名的首字母应该大写。如果类名由若干单词组成，那么每个单词的首字母应该大写，例如 MyFirstJavaClass 。
- 方法名：所有的方法名都应该以小写字母开头。如果方法名含有若干单词，则后面的每个单词首字母大写。
- 源文件名：源文件名必须和类名相同。当保存文件的时候，你应该使用类名作为文件名保存（切记Java是大小写敏感的），文件名的后缀为.java。（如果文件名和类名不相同则会导致编译错误）。
- 主方法入口：所有的Java 程序由**public static void main(String args[])**方法开始执行。

# Java标识符

Java所有的组成部分都需要名字。类名、变量名以及方法名都被称为标识符。

关于Java标识符，有以下几点需要注意：

- 所有的标识符都应该以字母（A-Z或者a-z）,美元符（$）、或者下划线（_）开始
- 首字符之后可以是任何字符的组合
- 关键字不能用作标识符
- 标识符是大小写敏感的
- 合法标识符举例：age、$salary、_value、__1_value
- 非法标识符举例：123abc、-salary

# Java修饰符

像其他语言一样，Java可以使用修饰符来修饰类中方法和属性。主要有两类修饰符：

- 可访问修饰符 : default, public , protected, private
- 不可访问修饰符 : final, abstract, strictfp

在后面的章节中我们会深入讨论Java修饰符。

# Java变量

Java中主要有如下几种类型的变量

- 局部变量
- 类变量（静态变量）
- 成员变量（非静态变量）

# Java数组

数组是储存在堆上的对象，可以保存多个同类型变量。在后面的章节中，我们将会学到如何声明、构造以及初始化一个数组。

# Java枚举

Java 5.0引入了枚举，枚举限制变量只能是预先设定好的值。使用枚举可以减少代码中的bug。

例如，我们为果汁店设计一个程序，它将限制果汁为小杯、中杯、大杯。这就意味着它不允许顾客点除了这三种尺寸外的果汁。

## 实例

```
class FreshJuice {
   enum FreshJuiceSize{ SMALL, MEDUIM, LARGE }
   FreshJuiceSize size;
}

public class FreshJuiceTest {
   public static void main(String args[]){
      FreshJuice juice = new FreshJuice();
      juice.size = FreshJuice. FreshJuiceSize.MEDUIM ;
   }
}
```

注意：枚举可以单独声明或者声明在类里面。方法、变量、构造函数也可以在枚举中定义。

# Java关键字

下面列出了Java保留字。这些保留字不能用于常量、变量、和任何标识符的名称。

| 关键字 | 描述 |
| --- | --- |
| abstract | 抽象方法，抽象类的修饰符 |
| assert | 断言条件是否满足 |
| boolean | 布尔数据类型 |
| break | 跳出循环或者label代码段 |
| byte | 8-bit 有符号数据类型 |
| | |

| | |
|---|---|
| case | switch语句的一个条件 |
| catch | 和try搭配扑捉异常信息 |
| char | 16-bit Unicode字符数据类型 |
| class | 定义类 |
| const | 未使用 |
| continue | 不执行循环体剩余部分 |
| default | switch语句中的默认分支 |
| do | 循环语句，循环体至少会执行一次 |
| double | 64-bit双精度浮点数 |
| else | if条件不成立时执行的分支 |
| enum | 枚举类型 |
| extends | 表示一个类是另一个类的子类 |
| final | 表示一个值在初始化之后就不能再改变了 表示方法不能被重写，或者一个类不能有子类 |
| finally | 为了完成执行的代码而设计的，主要是为了程序的健壮性和完整性，无论有没有异常发生都执行代码。 |
| float | 32-bit单精度浮点数 |
| for | for循环语句 |
| goto | 未使用 |
| if | 条件语句 |
| implements | 表示一个类实现了接口 |
| import | 导入类 |
| instanceof | 测试一个对象是否是某个类的实例 |
| int | 32位整型数 |
| interface | 接口，一种抽象的类型，仅有方法和常量的定义 |
| long | 64位整型数 |
| native | 表示方法用非java代码实现 |
| new | 分配新的类实例 |
| package | 一系列相关类组成一个包 |
| private | 表示私有字段，或者方法等，只能从类内部访问 |
| | |

| | |
|---|---|
| protected | 表示字段只能通过类或者其子类访问 子类或者在同一个包内的其他类 |
| public | 表示共有属性或者方法 |
| return | 方法返回值 |
| short | 16位数字 |
| static | 表示在类级别定义，所有实例共享的 |
| strictfp | 浮点数比较使用严格的规则 |
| super | 表示基类 |
| switch | 选择语句 |
| synchronized | 表示同一时间只能由一个线程访问的代码块 |
| this | 表示调用当前实例 或者调用另一个构造函数 |
| throw | 抛出异常 |
| throws | 定义方法可能抛出的异常 |
| transient | 修饰不要序列化的字段 |
| try | 表示代码块要做异常处理或者和finally配合表示是否抛出异常都执行finally中的代码 |
| void | 标记方法不返回任何值 |
| volatile | 标记字段可能会被多个线程同时访问，而不做同步 |
| while | while循环 |

# Java注释

类似于C/C++，Java也支持单行以及多行注释。注释中的字符将被Java编译器忽略。

```
public class MyFirstJavaProgram{
   /* 这是第一个Java程序
    *它将打印Hello World
    * 这是一个多行注释的示例
    */
    public static void main(String []args){
       // 这是单行注释的示例
       /* 这个也是单行注释的示例 */
       System.out.println("Hello World");
    }
}
```

# Java 空行

空白行，或者有注释的的行，Java编译器都会忽略掉。

## 继承

在Java中，一个类可以由其他类派生。如果你要创建一个类，而且已经存在一个类具有你所需要的属性或方法，那么你可以将新创建的类继承该类。

利用继承的方法，可以重用已存在类的方法和属性，而不用重写这些代码。被继承的类称为超类（super class），派生类称为子类（subclass）。

# 接口

在Java中，接口可理解为对象间相互通信的协议。接口在继承中扮演着很重要的角色。

接口只定义派生要用到的方法，但是方法的具体实现完全取决于派生类。

下一节介绍Java编程中的类和对象。之后你将会对Java中的类和对象有更清楚的认识。

# Java对象和类

Java作为一种面向对象语言。支持以下基本概念：

- 多态
- 继承
- 封装
- 抽象
- 类
- 对象
- 实例
- 方法
- 消息解析

本节我们重点研究对象和类的概念。

- 对象：对象是类的一个实例，有状态和行为。例如，一条狗是一个对象，它的状态有：颜色、名字、品种；行为有：摇尾巴、叫、吃等。
- 类：类是一个模板，它描述一类对象的行为和状态。

## Java中的对象

现在让我们深入了解什么是对象。看看周围真实的世界，会发现身边有很多对象，车，狗，人等等。所有这些对象都有自己的状态和行为。

拿一条狗来举例，它的状态有：名字、品种、颜色，行为有：叫、摇尾巴和跑。

对比现实对象和软件对象，它们之间十分相似。

软件对象也有状态和行为。软件对象的状态就是属性，行为通过方法体现。

在软件开发中，方法操作对象内部状态的改变，对象的相互调用也是通过方法来完成。

## Java中的类

类可以看成是创建Java对象的模板。

通过下面一个简单的类来理解下Java中类的定义：

```
public class Dog{
   String breed;
   int age;
   String color;
   void barking(){
   }

   void hungry(){
   }

   void sleeping(){
   }
}
```

一个类可以包含以下类型变量：

- 局部变量：在方法、构造方法或者语句块中定义的变量被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量就会自动销毁。
- 成员变量：成员变量是定义在类中，方法体之外的变量。这种变量在创建对象的时候实例化。成员变量可以被类中方法、构造方法和特定类的语句块访问。
- 类变量：类变量也声明在类中，方法体之外，但必须声明为static类型。

一个类可以拥有多个方法，在上面的例子中：barking()、hungry()和sleeping()都是Dog类的方法。

## 构造方法

每个类都有构造方法。如果没有显式地为类定义构造方法，Java编译器将会为该类提供一个默认构造方法。

在创建一个对象的时候，至少要调用一个构造方法。构造方法的名称必须与类同名，一个类可以有多个构造方法。

下面是一个构造方法示例：

```
public class Puppy{
   public Puppy(){
   }

   public Puppy(String name){
      // 这个构造器仅有一个参数：name
   }
}
```

## 创建对象

对象是根据类创建的。在Java中，使用关键字new来创建一个新的对象。创建对象需要以下三步：

- 声明：声明一个对象，包括对象名称和对象类型。
- 实例化：使用关键字new来创建一个对象。
- 初始化：使用new创建对象时，会调用构造方法初始化对象。

下面是一个创建对象的例子：

```
public class Puppy{
   public Puppy(String name){
      //这个构造器仅有一个参数：name
      System.out.println("Passed Name is :" + name );
   }
   public static void main(String []args){
      // 下面的语句将创建一个Puppy对象
      Puppy myPuppy = new Puppy( "tommy" );
   }
}
```

编译并运行上面的程序，会打印出下面的结果：

```
Passed Name is :tommy
```

## 访问实例变量和方法

通过已创建的对象来访问成员变量和成员方法，如下所示：

```
/* 实例化对象 */
ObjectReference = new Constructor();
/* 访问其中的变量 */
ObjectReference.variableName;
/* 访问类中的方法 */
ObjectReference.MethodName();
```

## 实例

下面的例子展示如何访问实例变量和调用成员方法：

```
public class Puppy{
   int puppyAge;
   public Puppy(String name){
      // 这个构造器仅有一个参数：name
      System.out.println("Passed Name is :" + name );
   }

   public void setAge( int age ){
       puppyAge = age;
   }

   public int getAge( ){
       System.out.println("Puppy's age is :" + puppyAge );
       return puppyAge;
   }

   public static void main(String []args){
      /* 创建对象 */
      Puppy myPuppy = new Puppy( "tommy" );
      /* 通过方法来设定age */
      myPuppy.setAge( 2 );
      /* 调用另一个方法获取age */
      myPuppy.getAge( );
      /*你也可以像下面这样访问成员变量 */
      System.out.println("Variable Value :" + myPuppy.puppyAge );
   }
}
```

编译并运行上面的程序，产生如下结果：

```
Passed Name is :tommy
Puppy's age is :2
Variable Value :2
```

# 源文件声明规则

在本节的最后部分，我们将学习源文件的声明规则。当在一个源文件中定义多个类，并且还有import语句和package语句时，要特别注意这些规则。

- 一个源文件中只能有一个public类
- 一个源文件可以有多个非public类
- 源文件的名称应该和public类的类名保持一致。例如：源文件中public类的类名是Employee，那么源文件应该命名为Employee.java。
- 如果一个类定义在某个包中，那么package语句应该在源文件的首行。
- 如果源文件包含import语句，那么应该放在package语句和类定义之间。如果没有package语句，那么import语句应该在源文件中最前面。
- import语句和package语句对源文件中定义的所有类都有效。在同一源文件

中，不能给不同的类不同的包声明。

类有若干种访问级别，并且类也分不同的类型：抽象类和final类等。这些将在访问控制章节介绍。

除了上面提到的几种类型，Java还有一些特殊的类，如：内部类、匿名类。

# Java包

包主要用来对类和接口进行分类。当开发Java程序时，可能编写成百上千的类，因此很有必要对类和接口进行分类。

# Import语句

在Java中，如果给出一个完整的限定名，包括包名、类名，那么Java编译器就可以很容易地定位到源代码或者类。Import语句就是用来提供一个合理的路径，使得编译器可以找到某个类。

例如，下面的命令行将会命令编译器载入java_installation/java/io路径下的所有类

```
import java.io.*;
```

# 一个简单的例子

在该例子中，我们创建两个类：Employee和EmployeeTest。

首先打开文本编辑器，把下面的代码粘贴进去。注意将文件保存为Employee.java。

Employee类有四个成员变量：name、age、designation和salary。该类显式声明了一个构造方法，该方法只有一个参数。

```
import java.io.*;
public class Employee{
   String name;
   int age;
   String designation;
   double salary;
   // Employee 类的构造器
   public Employee(String name){
      this.name = name;
   }
   // 设置age的值
   public void empAge(int empAge){
      age =  empAge;
   }
   /* 设置designation的值*/
   public void empDesignation(String empDesig){
      designation = empDesig;
   }
   /* 设置salary的值*/
   public void empSalary(double empSalary){
      salary = empSalary;
   }
   /* 打印信息 */
   public void printEmployee(){
      System.out.println("Name:"+ name );
      System.out.println("Age:" + age );
      System.out.println("Designation:" + designation );
      System.out.println("Salary:" + salary);
   }
}
```

程序都是从main方法开始执行。为了能运行这个程序，必须包含main方法并且创建一个实例对象。

下面给出EmployeeTest类，该类实例化2个Employee类的实例，并调用方法设置变量的值。

将下面的代码保存在EmployeeTest.java文件中。

```
import java.io.*;
public class EmployeeTest{

   public static void main(String args[]){
      /* 使用构造器创建两个对象 */
      Employee empOne = new Employee("James Smith");
      Employee empTwo = new Employee("Mary Anne");

      // 调用这两个对象的成员方法
      empOne.empAge(26);
      empOne.empDesignation("Senior Software Engineer");
      empOne.empSalary(1000);
      empOne.printEmployee();

      empTwo.empAge(21);
      empTwo.empDesignation("Software Engineer");
      empTwo.empSalary(500);
      empTwo.printEmployee();
   }
}
```

编译这两个文件并且运行EmployeeTest类，可以看到如下结果：

```
C :> javac Employee.java
C :> vi EmployeeTest.java
C :> javac  EmployeeTest.java
C :> java EmployeeTest
Name:James Smith
Age:26
Designation:Senior Software Engineer
Salary:1000.0
Name:Mary Anne
Age:21
Designation:Software Engineer
Salary:500.0
```

# Java基本数据类型

变量就是申请内存来存储值。也就是说，当创建变量的时候，需要在内存中申请空间。

内存管理系统根据变量的类型为变量分配存储空间，分配的空间只能用来储存该类型数据。

因此，通过定义不同类型的变量，可以在内存中储存整数、小数或者字符。

Java的两大数据类型:

- 内置数据类型
- 引用数据类型

## 内置数据类型

Java语言提供了八种基本类型。六种数字类型（四个整数型，两个浮点型），一种字符类型，还有一种布尔型。

byte：

- byte数据类型是8位、有符号的，以二进制补码表示的整数；
- 最小值是-128（-2^7）；
- 最大值是127（2^7-1）；
- 默认值是0；
- byte类型用在大型数组中节约空间，主要代替整数，因为byte变量占用的空间只有int类型的四分之一；
- 例子：byte a = 100，byte b = -50。

short：

- short数据类型是16位、有符号的以二进制补码表示的整数
- 最小值是-32768（-2^15）；
- 最大值是32767（2^15 - 1）；
- Short数据类型也可以像byte那样节省空间。一个short变量是int型变量所占空间的二分之一；
- 默认值是0；
- 例子：short s = 1000，short r = -20000。

int：

- int数据类型是32位、有符号的以二进制补码表示的整数；
- 最小值是-2,147,483,648（-2^31）；
- 最大值是2,147,485,647（2^31 - 1）；
- 一般地整型变量默认为int类型；
- 默认值是0；

- 例子：int a = 100000, int b = -200000。

long：

- long数据类型是64位、有符号的以二进制补码表示的整数；
- 最小值是-9,223,372,036,854,775,808（-2^63）；
- 最大值是9,223,372,036,854,775,807（2^63 -1）；
- 这种类型主要使用在需要比较大整数的系统上；
- 默认值是0L；
- 例子： long a = 100000L，int b = -200000L。

float：

- float数据类型是单精度、32位、符合IEEE 754标准的浮点数；
- float在储存大型浮点数组的时候可节省内存空间；
- 默认值是0.0f；
- 浮点数不能用来表示精确的值，如货币；
- 例子：float f1 = 234.5f。

double：

- double数据类型是双精度、64位、符合IEEE 754标准的浮点数；
- 浮点数的默认类型为double类型；
- double类型同样不能表示精确的值，如货币；
- 默认值是0.0f；
- 例子：double d1 = 123.4。

boolean：

- boolean数据类型表示一位的信息；
- 只有两个取值：true和false；
- 这种类型只作为一种标志来记录true/false情况；
- 默认值是false；
- 例子：boolean one = true。

char：

- char类型是一个单一的16位Unicode字符；
- 最小值是’\u0000’（即为0）；
- 最大值是’\uffff’（即为65,535）；
- char数据类型可以储存任何字符；
- 例子：char letter = ‘A’。

## 实例

对于数值类型的基本类型的取值范围，我们无需强制去记忆，因为它们的值都已经以常量的形式定义在对应的包装类中了。请看下面的例子：

```
public class PrimitiveTypeTest {
    public static void main(String[] args) {
```

```
        // byte
        System.out.println("基本类型：byte 二进制位数：" + Byte.SIZE);
        System.out.println("包装类：java.lang.Byte");
        System.out.println("最小值：Byte.MIN_VALUE=" + Byte.MIN_VALU
        System.out.println("最大值：Byte.MAX_VALUE=" + Byte.MAX_VALU
        System.out.println();

        // short
        System.out.println("基本类型：short 二进制位数：" + Short.SIZE
        System.out.println("包装类：java.lang.Short");
        System.out.println("最小值：Short.MIN_VALUE=" + Short.MIN_VA
        System.out.println("最大值：Short.MAX_VALUE=" + Short.MAX_VA
        System.out.println();

        // int
        System.out.println("基本类型：int 二进制位数：" + Integer.SIZE
        System.out.println("包装类：java.lang.Integer");
        System.out.println("最小值：Integer.MIN_VALUE=" + Integer.MI
        System.out.println("最大值：Integer.MAX_VALUE=" + Integer.MA
        System.out.println();

        // long
        System.out.println("基本类型：long 二进制位数：" + Long.SIZE);
        System.out.println("包装类：java.lang.Long");
        System.out.println("最小值：Long.MIN_VALUE=" + Long.MIN_VALU
        System.out.println("最大值：Long.MAX_VALUE=" + Long.MAX_VALU
        System.out.println();

        // float
        System.out.println("基本类型：float 二进制位数：" + Float.SIZE
        System.out.println("包装类：java.lang.Float");
        System.out.println("最小值：Float.MIN_VALUE=" + Float.MIN_VA
        System.out.println("最大值：Float.MAX_VALUE=" + Float.MAX_VA
        System.out.println();

        // double
        System.out.println("基本类型：double 二进制位数：" + Double.SI
        System.out.println("包装类：java.lang.Double");
        System.out.println("最小值：Double.MIN_VALUE=" + Double.MIN_
        System.out.println("最大值：Double.MAX_VALUE=" + Double.MAX_
        System.out.println();

        // char
        System.out.println("基本类型：char 二进制位数：" + Character.S
        System.out.println("包装类：java.lang.Character");
        // 以数值形式而不是字符形式将Character.MIN_VALUE输出到控制台
        System.out.println("最小值：Character.MIN_VALUE="
                + (int) Character.MIN_VALUE);
        // 以数值形式而不是字符形式将Character.MAX_VALUE输出到控制台
        System.out.println("最大值：Character.MAX_VALUE="
                + (int) Character.MAX_VALUE);
    }
}
```



编译以上代码输出结果如下所示：

```
基本类型：byte 二进制位数：8
包装类：java.lang.Byte
最小值：Byte.MIN_VALUE=-128
最大值：Byte.MAX_VALUE=127

基本类型：short 二进制位数：16
包装类：java.lang.Short
最小值：Short.MIN_VALUE=-32768
最大值：Short.MAX_VALUE=32767

基本类型：int 二进制位数：32
包装类：java.lang.Integer
最小值：Integer.MIN_VALUE=-2147483648
最大值：Integer.MAX_VALUE=2147483647

基本类型：long 二进制位数：64
包装类：java.lang.Long
最小值：Long.MIN_VALUE=-9223372036854775808
最大值：Long.MAX_VALUE=9223372036854775807

基本类型：float 二进制位数：32
包装类：java.lang.Float
最小值：Float.MIN_VALUE=1.4E-45
最大值：Float.MAX_VALUE=3.4028235E38

基本类型：double 二进制位数：64
包装类：java.lang.Double
最小值：Double.MIN_VALUE=4.9E-324
最大值：Double.MAX_VALUE=1.7976931348623157E308

基本类型：char 二进制位数：16
包装类：java.lang.Character
最小值：Character.MIN_VALUE=0
最大值：Character.MAX_VALUE=65535
```

Float和Double的最小值和最大值都是以科学记数法的形式输出的，结尾的"E+数字"表示E之前的数字要乘以10的多少倍。比如3.14E3就是3.14×1000=3140，3.14E-3就是3.14/1000=0.00314。

实际上，JAVA中还存在另外一种基本类型void，它也有对应的包装类java.lang.Void，不过我们无法直接对它们进行操作。

# 引用类型

- 引用类型变量由类的构造函数创建，可以使用它们访问所引用的对象。这些变

量在声明时被指定为一个特定的类型，比如Employee、Pubby等。变量一旦声明后，类型就不能被改变了。
- 对象、数组都是引用数据类型。
- 所有引用类型的默认值都是null。
- 一个引用变量可以用来引用与任何与之兼容的类型。
- 例子：Animal animal = new Animal("giraffe")。

# Java常量

常量就是一个固定值。它们不需要计算，直接代表相应的值。

常量指不能改变的量。 在Java中用final标志，声明方式和变量类似：

```
final double PI = 3.1415927;
```

虽然常量名也可以用小写，但为了便于识别，通常使用大写字母表示常量。

字面量可以赋给任何内置类型的变量。例如：

```
byte a = 68;
char a = 'A'
```

byte、int、long、和short都可以用十进制、16进制以及8进制的方式来表示。

当使用常量的时候，前缀o表明是8进制，而前缀0x代表16进制。例如：

```
int decimal = 100;
int octal = 0144;
int hexa =  0x64;
```

和其他语言一样，Java的字符串常量也是包含在两个引号之间的字符序列。下面是字符串型字面量的例子：

```
"Hello World"
"two\nlines"
"\"This is in quotes\""
```

字符串常量和字符常量都可以包含任何Unicode字符。例如：

```
char a = '\u0001';
String a = "\u0001";
```

Java语言支持一些特殊的转义字符序列。

| 符号 | 字符含义 |
|---|---|
| \n | 换行 (0x0a) |
| \r | 回车 (0x0d) |
| \f | 换页符(0x0c) |
| \b | 退格 (0x08) |
| \s | 空格 (0x20) |
| \t | 制表符 |
| \" | 双引号 |
| \' | 单引号 |
| \ | 反斜杠 |
| \ddd | 八进制字符 (ddd) |
| \uxxxx | 16进制Unicode字符 (xxxx) |

这一节讲解了Java的基本数据类型。下一节将探讨不同的变量类型以及它们的用法。

# Java变量类型

在Java语言中，所有的变量在使用前必须声明。声明变量的基本格式如下：

```
type identifier [ = value][, identifier [= value] ...] ;
```

格式说明：type为Java数据类型。identifier是变量名。可以使用逗号隔开来声明多个同类型变量。

以下列出了一些变量的声明实例。注意有些包含了初始化过程。

```
int a, b, c;         // 声明三个int型整数：a、 b、c。
int d = 3, e, f = 5; // d声明三个整数并赋予初值。
byte z = 22;         // 声明并初始化z。
double pi = 3.14159; // 声明了pi。
char x = 'x';        // 变量x的值是字符'x'。
```

Java语言支持的变量类型有：

- 局部变量
- 成员变量
- 类变量

# Java局部变量

- 局部变量声明在方法、构造方法或者语句块中；
- 局部变量在方法、构造方法、或者语句块被执行的时候创建，当它们执行完成后，变量将会被销毁；
- 访问修饰符不能用于局部变量；
- 局部变量只在声明它的方法、构造方法或者语句块中可见；
- 局部变量是在栈上分配的。
- 局部变量没有默认值，所以局部变量量被声明后，必须经过初始化，才可以使用。

## 实例1

在以下实例中age是一个局部变量。定义在pubAge()方法中，它的作用域就限制在这个方法中。

```
public class Test{
   public void pupAge(){
      int age = 0;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }

   public static void main(String args[]){
      Test test = new Test();
      test.pupAge();
   }
}
```

以上实例编译运行结果如下：

```
Puppy age is: 7
```

## 实例2

在下面的例子中age变量没有初始化，所以在编译时出错。

```
public class Test{
   public void pupAge(){
      int age;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }

   public static void main(String args[]){
      Test test = new Test();
      test.pupAge();
   }
}
```

以上实例编译运行结果如下:

```
Test.java:4:variable number might not have been initialized
age = age + 7;
         ^
1 error
```

# 实例变量

- 实例变量声明在一个类中，但在方法、构造方法和语句块之外；

- 当一个对象被实例化之后，每个实例变量的值就跟着确定；
- 实例变量在对象创建的时候创建，在对象被销毁的时候销毁；
- 实例变量的值应该至少被一个方法、构造方法或者语句块引用，使得外部能够通过这些方式获取实例变量信息；
- 实例变量可以声明在使用前或者使用后；
- 访问修饰符可以修饰实例变量；
- 实例变量对于类中的方法、构造方法或者语句块是可见的。一般情况下应该把实例变量设为私有。通过使用访问修饰符可以使实例变量对子类可见；
- 实例变量具有默认值。数值型变量的默认值是0，布尔型变量的默认值是false，引用类型变量的默认值是null。变量的值可以在声明时指定，也可以在构造方法中指定；
- 实例变量可以直接通过变量名访问。但在静态方法以及其他类中，就应该使用完全限定名：ObejectReference.VariableName。

实例：

```
import java.io.*;
public class Employee{
   // 这个成员变量对子类可见
   public String name;
   // 私有变量，仅在该类可见
   private double salary;
   //在构造器中对name赋值
   public Employee (String empName){
      name = empName;
   }
   //设定salary的值
   public void setSalary(double empSal){
      salary = empSal;
   }
   // 打印信息
   public void printEmp(){
      System.out.println("name  : " + name );
      System.out.println("salary :" + salary);
   }

   public static void main(String args[]){
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
      empOne.printEmp();
   }
}
```

以上实例编译运行结果如下:

```
name  : Ransika
salary :1000.0
```

# 类变量（静态变量）

- 类变量也称为静态变量，在类中以static关键字声明，但必须在方法构造方法和语句块之外。
- 无论一个类创建了多少个对象，类只拥有类变量的一份拷贝。
- 静态变量除了被声明为常量外很少使用。常量是指声明为publc/private，final和static类型的变量。常量初始化后不可改变。
- 静态变量储存在静态存储区。经常被声明为常量，很少单独使用static声明变量。
- 静态变量在程序开始时创建，在程序结束时销毁。
- 与实例变量具有相似的可见性。但为了对类的使用者可见，大多数静态变量声明为public类型。
- 默认值和实例变量相似。数值型变量默认值是0，布尔型默认值是false，引用类型默认值是null。变量的值可以在声明的时候指定，也可以在构造方法中指定。此外，静态变量还可以在静态语句块中初始化。
- 静态变量可以通过：*ClassName.VariableName*的方式访问。
- 类变量被声明为public static final类型时，类变量名称必须使用大写字母。如果静态变量不是public和final类型，其命名方式与实例变量以及局部变量的命名方式一致。

实例：

```
import java.io.*;
public class Employee{
   //salary是静态的私有变量
   private static double salary;
   // DEPARTMENT是一个常量
   public static final String DEPARTMENT = "Development ";
   public static void main(String args[]){
      salary = 1000;
      System.out.println(DEPARTMENT+"average salary:"+salary);
   }
}
```

以上实例编译运行结果如下:

```
Development average salary:1000
```

注意：如果其他类想要访问该变量，可以这样访问：Employee.DEPARTMENT。

本章节中我们学习了Java的变量类型，下一章节中我们将介绍Java修饰符的使用。

# Java修饰符

Java语言提供了很多修饰符，主要分为以下两类：

- 访问修饰符
- 非访问修饰符

修饰符用来定义类、方法或者变量，通常放在语句的最前端。我们通过下面的例子来说明：

```
public class className {
   // ...
}
private boolean myFlag;
static final double weeks = 9.5;
protected static final int BOXWIDTH = 42;
public static void main(String[] arguments) {
   // 方法体
}
```

## 访问控制修饰符

Java中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。Java支持4种不同的访问权限。

默认的，也称为default，在同一包内可见，不使用任何修饰符。

私有的，以private修饰符指定，在同一类内可见。

共有的，以public修饰符指定，对所有类可见。

受保护的，以protected修饰符指定，对同一包内的类和所有子类可见。

### 默认访问修饰符-不使用任何关键字

使用默认访问修饰符声明的变量和方法，对同一个包内的类是可见的。接口里的变量都隐式声明为public static final,而接口里的方法默认情况下访问权限为public。

实例：

如下例所示，变量和方法的声明可以不使用任何修饰符。

```
String version = "1.5.1";
boolean processOrder() {
   return true;
}
```

## 私有访问修饰符-private

私有访问修饰符是最严格的访问级别，所以被声明为private的方法、变量和构造方法只能被所属类访问，并且类和接口不能声明为private。

声明为私有访问类型的变量只能通过类中公共的getter方法被外部类访问。

Private访问修饰符的使用主要用来隐藏类的实现细节和保护类的数据。

下面的类使用了私有访问修饰符：

```
public class Logger {
   private String format;
   public String getFormat() {
      return this.format;
   }
   public void setFormat(String format) {
      this.format = format;
   }
}
```

实例中，Logger类中的format变量为私有变量，所以其他类不能直接得到和设置该变量的值。为了使其他类能够操作该变量，定义了两个public方法：getFormat()（返回format的值）和setFormat(String)（设置format的值）

## 公有访问修饰符-public

被声明为public的类、方法、构造方法和接口能够被任何其他类访问。

如果几个相互访问的public类分布在不用的包中，则需要导入相应public类所在的包。由于类的继承性，类所有的公有方法和变量都能被其子类继承。

以下函数使用了公有访问控制：

```
public static void main(String[] arguments) {
   // ...
}
```

Java程序的main() 方法必须设置成公有的，否则，Java解释器将不能运行该类。

## 受保护的访问修饰符-protected

被声明为protected的变量、方法和构造器能被同一个包中的任何其他类访问，也能够被不同包中的子类访问。

Protected访问修饰符不能修饰类和接口，方法和成员变量能够声明为protected，但是接口的成员变量和成员方法不能声明为protected。

子类能访问Protected修饰符声明的方法和变量，这样就能保护不相关的类使用这些方法和变量。

下面的父类使用了protected访问修饰符，子类重载了父类的openSpeaker()方法。

```
class AudioPlayer {
   protected boolean openSpeaker(Speaker sp) {
      // 实现细节
   }
}

class StreamingAudioPlayer {
   boolean openSpeaker(Speaker sp) {
      // 实现细节
   }
}
```

如果把openSpeaker()方法声明为private，那么除了AudioPlayer之外的类将不能访问该方法。如果把openSpeaker()声明为public，那么所有的类都能够访问该方法。如果我们只想让该方法对其所在类的子类可见，则将该方法声明为protected。

### 访问控制和继承

请注意以下方法继承的规则：

- 父类中声明为public的方法在子类中也必须为public。
- 父类中声明为protected的方法在子类中要么声明为protected，要么声明为public。不能声明为private。
- 父类中默认修饰符声明的方法，能够在子类中声明为private。
- 父类中声明为private的方法，不能够被继承。

## 非访问修饰符

为了实现一些其他的功能，Java也提供了许多非访问修饰符。

static修饰符，用来创建类方法和类变量。

Final修饰符，用来修饰类、方法和变量，final修饰的类不能够被继承，修饰的方法不能被继承类重新定义，修饰的变量为常量，是不可修改的。

Abstract修饰符，用来创建抽象类和抽象方法。

Synchronized和volatile修饰符，主要用于线程的编程。

## Static修饰符

- 静态变量：

  Static关键字用来声明独立于对象的静态变量，无论一个类实例化多少对象，它的静态变量只有一份拷贝。 静态变量也被成为类变量。局部变量能被声明为static变量。

- 静态方法：

  Static关键字用来声明独立于对象的静态方法。静态方法不能使用类的非静态变量。静态方法从参数列表得到数据，然后计算这些数据。

对类变量和方法的访问可以直接使用classname.variablename和classname.methodname的方式访问。

如下例所示，static修饰符用来创建类方法和类变量。

```
public class InstanceCounter {
   private static int numInstances = 0;
   protected static int getCount() {
      return numInstances;
   }

   private static void addInstance() {
      numInstances++;
   }

   InstanceCounter() {
      InstanceCounter.addInstance();
   }

   public static void main(String[] arguments) {
      System.out.println("Starting with " +
      InstanceCounter.getCount() + " instances");
      for (int i = 0; i < 500; ++i){
         new InstanceCounter();
          }
      System.out.println("Created " +
      InstanceCounter.getCount() + " instances");
   }
}
```

以上实例运行编辑结果如下:

```
Started with 0 instances
Created 500 instances
```

## Final修饰符

**Final变量：**

Final变量能被显式地初始化并且只能初始化一次。被声明为final的对象的引用不能指向不同的对象。但是final对象里的数据可以被改变。也就是说final对象的引用不能改变，但是里面的值可以改变。

Final修饰符通常和static修饰符一起使用来创建类常量。

实例:

```
public class Test{
  final int value = 10;
  // 下面是声明常量的实例
  public static final int BOXWIDTH = 6;
  static final String TITLE = "Manager";

  public void changeValue(){
     value = 12; //将输出一个错误
  }
}
```

## Final方法

类中的Final方法可以被子类继承，但是不能被子类修改。

声明final方法的主要目的是防止该方法的内容被修改。

如下所示，使用final修饰符声明方法。

```
public class Test{
    public final void changeName(){
       // 方法体
    }
}
```

## Final类

Final类不能被继承，没有类能够继承final类的任何特性。

实例：

```
public final class Test {
   // 类体
}
```

## Abstract修饰符

抽象类：

抽象类不能用来实例化对象，声明抽象类的唯一目的是为了将来对该类进行扩充。

一个类不能同时被abstract和final修饰。如果一个类包含抽象方法，那么该类一定要声明为抽象类，否则将出现编译错误。

抽象类可以包含抽象方法和非抽象方法。

实例：

```
abstract class Caravan{
   private double price;
   private String model;
   private String year;
   public abstract void goFast(); //抽象方法
   public abstract void changeColor();
}
```

## 抽象方法

抽象方法是一种没有任何实现的方法，该方法的的具体实现由子类提供。抽象方法不能被声明成final和strict。

任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类。

如果一个类包含若干个抽象方法，那么该类必须声明为抽象类。抽象类可以不包含抽象方法。

抽象方法的声明以分号结尾，例如：public abstract sample();

实例：

```
public abstract class SuperClass{
    abstract void m(); //抽象方法
}

class SubClass extends SuperClass{
     //实现抽象方法
      void m(){
          .........
      }
}
```

## Synchronized修饰符

Synchronized关键字声明的方法同一时间只能被一个线程访问。Synchronized修饰符可以应用于四个访问修饰符。

实例：

```
public synchronized void showDetails(){
.......
}
```

## Transient修饰符

序列化的对象包含被transient修饰的实例变量时，java虚拟机(JVM)跳过该特定的变量。

该修饰符包含在定义变量的语句中，用来预处理类和变量的数据类型。

实例：

```
public transient int limit = 55;   // will not persist
public int b; // will persist
```

## volatile修饰符

Volatile修饰的成员变量在每次被线程访问时，都强迫从共享内存中重读该成员变量的值。而且，当成员变量发生变化时，强迫线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的同一个值。一个volatile对象引用可能是null。

实例：

```
public class MyRunnable implements Runnable
{
    private volatile boolean active;
    public void run()
    {
        active = true;
        while (active) // line 1
        {
            // 代码
        }
    }
    public void stop()
    {
        active = false; // line 2
    }
}
```

一般地，在一个线程中调用run()方法，在另一个线程中调用stop()方法。如果line 1中的active位于缓冲区的值被使用，那么当把line 2中的active设置成false时，循环也不会停止。

# Java运算符

计算机的最基本用途之一就是执行数学运算，作为一门计算机语言，Java也提供了一套丰富的运算符来操纵变量。我们可以把运算符分成以下几组：

- 算术运算符
- 关系运算符
- 位运算符
- 逻辑运算符
- 赋值运算符
- 其他运算符

## 算术运算符

算术运算符用在数学表达式中，它们的作用和在数学中的作用一样。下表列出了所有的算术运算符。

表格中的实例假设整数变量A的值为10，变量B的值为20：

| 操作符 | 描述 | 例子 |
| --- | --- | --- |
| + | 加法 - 相加运算符两侧的值 | A + B等于30 |
| - | 减法 - 左操作数减去右操作数 | A – B等于-10 |
| * | 乘法 - 相乘操作符两侧的值 | A * B等于200 |
| / | 除法 - 左操作数除以右操作数 | B / A等于2 |
| % | 取模 - 右操作数除左操作数的余数 | B%A等于0 |
| + + | 自增 - 操作数的值增加1 | B + +等于21 |
| - | 自减 - 操作数的值减少1 | B - -等于19 |

### 实例

下面的简单示例程序演示了算术运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

```
public class Test {

  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     int c = 25;
     int d = 25;
     System.out.println("a + b = " + (a + b) );
     System.out.println("a - b = " + (a - b) );
     System.out.println("a * b = " + (a * b) );
     System.out.println("b / a = " + (b / a) );
     System.out.println("b % a = " + (b % a) );
     System.out.println("c % a = " + (c % a) );
     System.out.println("a++   = " +  (a++) );
     System.out.println("b--   = " +  (a--) );
     // Check the difference in d++ and ++d
     System.out.println("d++   = " +  (d++) );
     System.out.println("++d   = " +  (++d) );
  }
}
```

以上实例编译运行结果如下：

```
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
b--   = 11
d++   = 25
++d   = 27
```

# 关系运算符

下表为Java支持的关系运算符

表格中的实例整数变量A的值为10，变量B的值为20：

| 运算符 | 描述 | 例子 |
| --- | --- | --- |
| == | 检查如果两个操作数的值是否相等，如果相等则条件为真。 | （A == B）为假(非真)。 |
| =! | 检查如果两个操作数的值是否相等，如果值不相等则条件为真。 | （A =! B）为真。 |
| > | 检查左操作数的值是否大于右操作数的值，如果是那么条件为真。 | （A > B）非真。 |
| < | 检查左操作数的值是否小于右操作数的值，如果是那么条件为真。 | （A < B）为真。 |
| >= | 检查左操作数的值是否大于或等于右操作数的值，如果是那么条件为真。 | （A >= B）为假。 |
| <= | 检查左操作数的值是否小于或等于右操作数的值，如果是那么条件为真。 | （A <= B）为真。 |

## 实例

下面的简单示例程序演示了关系运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

```
public class Test {

  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     System.out.println("a == b = " + (a == b) );
     System.out.println("a != b = " + (a != b) );
     System.out.println("a > b = " + (a > b) );
     System.out.println("a < b = " + (a < b) );
     System.out.println("b >= a = " + (b >= a) );
     System.out.println("b <= a = " + (b <= a) );
  }
}
```

以上实例编译运行结果如下：

```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

# 位运算符

Java定义了位运算符，应用于整数类型(int)，长整型(long)，短整型(short)，字符型(char)，和字节型(byte)等类型。

位运算符作用在所有的位上，并且按位运算。假设a = 60，和b = 13;它们的二进制格式表示将如下：

```
A = 0011 1100
B = 0000 1101
-----------------
A&b = 0000 1100
A | B = 0011 1101
^ B = 0011 0001
~A= 1100 0011
```

下表列出了位运算符的基本运算,假设整数变量A的值为60和变量B的值为13：

| 操作符 | 描述 | 例子 |
|---|---|---|
| & | 按位与操作符，当且仅当两个操作数的某一位都非0时候结果的该位才为1。 | （A&B），得到12，即0000 1100 |
| \| | 按位或操作符，只要两个操作数的某一位有一个非0时候结果的该位就为1。 | （A \| B）得到61，即 0011 1101 |
| ^ | 按位异或操作符，两个操作数的某一位不相同时候结果的该位就为1。 | （A ^ B）得到49，即 0011 0001 |
| ? | 按位补运算符翻转操作数的每一位。 | （?A）得到-60，即1100 0011 |
| << | 按位左移运算符。左操作数按位左移右操作数指定的位数。 | A << 2得到240，即 1111 0000 |
| >> | 按位右移运算符。左操作数按位右移右操作数指定的位数。 | A >> 2得到15即 1111 |
| >>> | 按位右移补零操作符。左操作数的值按右操作数指定的位数右移，移动得到的空位以零填充。 | A>>>2得到15即0000 1111 |

## 实例

下面的简单示例程序演示了位运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

```
public class Test {
  public static void main(String args[]) {
     int a = 60; /* 60 = 0011 1100 */
     int b = 13; /* 13 = 0000 1101 */
     int c = 0;
     c = a & b;       /* 12 = 0000 1100 */
     System.out.println("a & b = " + c );

     c = a | b;       /* 61 = 0011 1101 */
     System.out.println("a | b = " + c );

     c = a ^ b;       /* 49 = 0011 0001 */
     System.out.println("a ^ b = " + c );

     c = ~a;          /*-61 = 1100 0011 */
     System.out.println("~a = " + c );

     c = a << 2;     /* 240 = 1111 0000 */
     System.out.println("a << 2 = " + c );

     c = a >> 2;     /* 215 = 1111 */
     System.out.println("a >> 2  = " + c );

     c = a >>> 2;     /* 215 = 0000 1111 */
     System.out.println("a >>> 2 = " + c );
  }
}
```

以上实例编译运行结果如下：

```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 15
a >>> 15
```

# 逻辑运算符

下表列出了逻辑运算符的基本运算，假设布尔变量A为真，变量B为假

| 操作符 | 描述 | 例子 |
| --- | --- | --- |
| && | 称为逻辑与运算符。当且仅当两个操作数都为真，条件才为真。 | （A && B）为假。 |
| \|\| | 称为逻辑或操作符。如果任何两个操作数任何一个为真，条件为真。 | （A \|\| B）为真。 |
| ! | 称为逻辑非运算符。用来反转操作数的逻辑状态。如果条件为true，则逻辑非运算符将得到false。 | !（A && B）为真。 |

## 实例

下面的简单示例程序演示了逻辑运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

```
public class Test {
  public static void main(String args[]) {
     boolean a = true;
     boolean b = false;
     System.out.println("a && b = " + (a&&b));
     System.out.println("a || b = " + (a||b) );
     System.out.println("!(a && b) = " + !(a && b));
  }
}
```

以上实例编译运行结果如下：

```
a && b = false
a || b = true
!(a && b) = true
```

# 赋值运算符

下面是Java语言支持的赋值运算符：

| 操作符 | 描述 | 例子 |
|---|---|---|
| = | 简单的赋值运算符，将右操作数的值赋给左侧操作数 | C = A + B将把A + B得到的值赋给C |
| += | 加和赋值操作符，它把左操作数和右操作数相加赋值给左操作数 | C += A等价于C = C + A |
| -= | 减和赋值操作符，它把左操作数和右操作数相减赋值给左操作数 | C -= A等价于C = C - A |
| *= | 乘和赋值操作符，它把左操作数和右操作数相乘赋值给左操作数 | C = *A*等价于*C* = *C* A |
| /= | 除和赋值操作符，它把左操作数和右操作数相除赋值给左操作数 | C /= A等价于C = C / A |
| %= | 取模和赋值操作符，它把左操作数和右操作数取模后赋值给左操作数 | C %= A等价于C = C %A |
| <<= | 左移位赋值运算符 | C < = 2等价于C = C << 2 |
| >>= | 右移位赋值运算符 | C >>= 2等价于C = C >> 2 |
| &= | 按位与赋值运算符 | C &= 2等价于C = C& 2 |
| ^= | 按位异或赋值操作符 | C ^= 2等价于C = C ^ 2 |
| \|= | 按位或赋值操作符 | C \|= 2等价于C = C \| 2 |

## 实例

面的简单示例程序演示了赋值运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

```
public class Test {
  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     int c = 0;
     c = a + b;
     System.out.println("c = a + b = " + c );
     c += a ;
     System.out.println("c += a  = " + c );
     c -= a ;
     System.out.println("c -= a = " + c );
     c *= a ;
     System.out.println("c *= a = " + c );
     a = 10;
     c = 15;
     c /= a ;
     System.out.println("c /= a = " + c );
     a = 10;
     c = 15;
     c %= a ;
     System.out.println("c %= a  = " + c );
     c <<= 2 ;
     System.out.println("c <<= 2 = " + c );
     c >>= 2 ;
     System.out.println("c >>= 2 = " + c );
     c >>= 2 ;
     System.out.println("c >>= a = " + c );
     c &= a ;
     System.out.println("c &= 2  = " + c );
     c ^= a ;
     System.out.println("c ^= a   = " + c );
     c |= a ;
     System.out.println("c |= a   = " + c );
  }
}
```

以上实例编译运行结果如下：

```
c = a + b = 30
c += a  = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a  = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a  = 0
c ^= a   = 10
c |= a   = 10
```

# 条件运算符（?:）

条件运算符也被称为三元运算符。该运算符有3个操作数，并且需要判断布尔表达式的值。该运算符的主要是决定哪个值应该赋值给变量。

```
variable x = (expression) ? value if true : value if false
```

## 实例

```
public class Test {
   public static void main(String args[]){
      int a , b;
      a = 10;
b = (a == 1) ? 20: 30;
System.out.println( "Value of b is : " +  b );
      b = (a == 10) ? 20: 30;
     System.out.println( "Value of b is : " + b );
   }
}
```

以上实例编译运行结果如下：

```
Value of b is : 30
Value of b is : 20
```

# instanceOf 运算符

该运算符用于操作对象实例，检查该对象是否是一个特定类型（类类型或接口类型）。

instanceof运算符使用格式如下：

```
( Object reference variable ) instanceOf  (class/interface type)
```

如果运算符左侧变量所指的对象，是操作符右侧类或接口(class/interface)的一个对象，那么结果为真。

下面是一个例子：

```
String name = 'James';
boolean result = name instanceOf String; // 由于name是Strine类型，所以
```

如果被比较的对象兼容于右侧类型,该运算符仍然返回true。

看下面的例子：

```
class Vehicle {}

public class Car extends Vehicle {
   public static void main(String args[]){
      Vehicle a = new Car();
      boolean result =  a instanceof Car;
      System.out.println( result);
   }
}
```

以上实例编译运行结果如下：

```
true
```

# Java运算符优先级

当多个运算符出现在一个表达式中，谁先谁后呢？这就涉及到运算符的优先级别的问题。在一个多运算符的表达式中，运算符优先级不同会导致最后得出的结果差别甚大。

例如，（1+3）＋（3+2）*2，这个表达式如果按加号最优先计算，答案就是 18，如果按照乘号最优先，答案则是 14。

再如，x = 7 + 3 *2;这里x得到13*，而不是*20*，因为乘法运算符比加法运算符有较高的优先级，所以先计算3* 2得到6，然后再加7。

下表中具有最高优先级的运算符在的表的最上面，最低优先级的在表的底部。

| 类别 | 操作符 | 关联性 | | |
|---|---|---|---|---|
| 后缀 | () [] . (点操作符) | 左到右 | | |
| 一元 | + + - ！? | 从右到左 | | |
| 乘性 | * /% | 左到右 | | |
| 加性 | + - | 左到右 | | |
| 移位 | >> >>> << | 左到右 | | |
| 关系 | >> = << = | 左到右 | | |
| 相等 | == != | 左到右 | | |
| 按位与 | ＆ | 左到右 | | |
| 按位异或 | ^ | 左到右 | | |
| 按位或 | | | 左到右 | |
| 逻辑与 | && | 左到右 | | |
| 逻辑或 | | | | 左到右 |
| 条件 | ？： | 从右到左 | | |
| 赋值 | = + = - = * = / =%= >> = << =&= ^ = | = | 从右到左 | |
| 逗号 | ， | 左到右 | | |

# Java循环结构 - for, while 及 do...while

顺序结构的程序语句只能被执行一次。如果您想要同样的操作执行多次,，就需要使用循环结构。

Java中有三种主要的循环结构：

- while循环
- do…while循环
- for循环

在Java5中引入了一种主要用于数组的增强型for循环。

## while循环

while是最基本的循环，它的结构为：

```
while( 布尔表达式 ) {
    //循环内容
}
```

只要布尔表达式为true，循环体会一直执行下去。

### 实例

```
public class Test {
   public static void main(String args[]) {
      int x = 10;
      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```

以上实例编译运行结果如下：

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

## do…while循环

对于while语句而言，如果不满足条件，则不能进入循环。但有时候我们需要即使不满足条件，也至少执行一次。

do…while循环和while循环相似，不同的是，do…while循环至少会执行一次。

```
do {
       //代码语句
}while(布尔表达式);
```

注意：布尔表达式在循环体的后面，所以语句块在检测布尔表达式之前已经执行了。 如果布尔表达式的值为true，则语句块一直执行，直到布尔表达式的值为false。

### 实例

```
public class Test {

   public static void main(String args[]){
      int x = 10;

      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

以上实例编译运行结果如下：

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

## for循环

虽然所有循环结构都可以用while或者do...while表示，但Java提供了另一种语句 —— for循环，使一些循环结构变得更加简单。

for循环执行的次数是在执行前就确定的。语法格式如下

```
for(初始化; 布尔表达式; 更新) {
    //代码语句
}
```

关于for循环有以下几点说明：

- 最先执行初始化步骤。可以声明并初始化一个或多个循环控制变量，也可以是空语句。
- 然后，检测布尔表达式的值。如果为true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。
- 执行一次循环后，更新循环控制变量。
- 再次检测布尔表达式。循环执行上面的过程。

## 实例

```
public class Test {

   public static void main(String args[]) {

      for(int x = 10; x < 20; x = x+1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
}
```

以上实例编译运行结果如下：

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

# Java增强for循环

Java5引入了一种主要用于数组的增强型for循环。

Java增强for循环语法格式如下:

```
for(声明语句 : 表达式)
{
   //代码句子
}
```

声明语句：声明新的局部变量，该变量的类型必须和数组元素的类型匹配。其作用域限定在循环语句块，其值与此时数组元素的值相等。

表达式：表达式是要访问的数组名，或者是返回值为数组的方法。

## 实例

```
public class Test {

   public static void main(String args[]){
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ){
         System.out.print( x );
         System.out.print(",");
      }
      System.out.print("\n");
      String [] names ={"James", "Larry", "Tom", "Lacy"};
      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

以上实例编译运行结果如下：

```
10,20,30,40,50,
James,Larry,Tom,Lacy,
```

# break关键字

break主要用在循环语句或者switch语句中，用来跳出整个语句块。

break跳出最里层的循环，并且继续执行该循环下面的语句。

## 语法

break的用法很简单，就是循环结构中的一条语句：

```
break;
```

## 实例

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

以上实例编译运行结果如下：

```
10
20
```

# continue关键字

continue适用于任何循环控制结构中。作用是让程序立刻跳转到下一次循环的迭代。

在for循环中，continue语句使程序立即跳转到更新语句。

在while或者do…while循环中，程序立即跳转到布尔表达式的判断语句。

## 语法

continue就是循环体中一条简单的语句：

```
continue;
```

## 实例

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          continue;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

以上实例编译运行结果如下：

```
10
20
40
50
```

# Java分支结构 - if...else/switch

顺序结构只能顺序执行，不能进行判断和选择，因此需要分支结构。

Java有两种分支结构：

- if语句
- switch语句

## if语句

一个if语句包含一个布尔表达式和一条或多条语句。

### 语法

If语句的用语法如下：

```
if(布尔表达式)
{
   //如果布尔表达式为true将执行的语句
}
```

如果布尔表达式的值为true，则执行if语句中的代码块。否则执行If语句块后面的代码。

```
public class Test {

   public static void main(String args[]){
      int x = 10;

      if( x < 20 ){
         System.out.print("这是 if 语句");
      }
   }
}
```

以上代码编译运行结果如下：

```
这是 if 语句
```

## if...else语句

if语句后面可以跟else语句，当if语句的布尔表达式值为false时，else语句块会被执行。

## 语法

if…else的用法如下：

```
if(布尔表达式){
   //如果布尔表达式的值为true
}else{
   //如果布尔表达式的值为false
}
```

## 实例

```
public class Test {

   public static void main(String args[]){
      int x = 30;

      if( x < 20 ){
         System.out.print("这是 if 语句");
      }else{
         System.out.print("这是 else 语句");
      }
   }
}
```

以上代码编译运行结果如下：

```
这是 else 语句
```

# if...else if...else语句

if语句后面可以跟elseif…else语句，这种语句可以检测到多种可能的情况。

使用if，else if，else语句的时候，需要注意下面几点：

- if语句至多有1个else语句，else语句在所有的elseif语句之后。
- If语句可以有若干个elseif语句，它们必须在else语句之前。
- 一旦其中一个else if语句检测为true，其他的else if以及else语句都将跳过执行。

## 语法

if...else语法格式如下:

```
if(布尔表达式 1){
   //如果布尔表达式 1的值为true执行代码
}else if(布尔表达式 2){
   //如果布尔表达式 2的值为true执行代码
}else if(布尔表达式 3){
   //如果布尔表达式 3的值为true执行代码
}else {
   //如果以上布尔表达式都不为true执行代码
}
```

### 实例

```
public class Test {

   public static void main(String args[]){
      int x = 30;

      if( x == 10 ){
         System.out.print("Value of X is 10");
      }else if( x == 20 ){
         System.out.print("Value of X is 20");
      }else if( x == 30 ){
         System.out.print("Value of X is 30");
      }else{
         System.out.print("This is else statement");
      }
   }
}
```

以上代码编译运行结果如下：

```
Value of X is 30
```

## 嵌套的if…else语句

使用嵌套的if-else语句是合法的。也就是说你可以在另一个if或者elseif语句中使用if或者elseif语句。

### 语法

嵌套的if…else语法格式如下：

```
if(布尔表达式 1){
   ////如果布尔表达式 1的值为true执行代码
   if(布尔表达式 2){
      ////如果布尔表达式 2的值为true执行代码
   }
}
```

你可以像 *if* 语句一样嵌套 *else if...else*。

### 实例

```
public class Test {

   public static void main(String args[]){
      int x = 30;
      int y = 10;

      if( x == 30 ){
         if( y == 10 ){
             System.out.print("X = 30 and Y = 10");
          }
       }
    }
}
```

以上代码编译运行结果如下：

```
X = 30 and Y = 10
```

## switch语句

switch语句判断一个变量与一系列值中某个值是否相等，每个值称为一个分支。

### 语法

switch语法格式如下：

```
switch(expression){
    case value :
       //语句
       break; //可选
    case value :
       //语句
       break; //可选
    //你可以有任意数量的case语句
    default : //可选
       //语句
}
```

switch语句有如下规则：

- switch语句中的变量类型只能为byte、short、int或者char。
- switch语句可以拥有多个case语句。每个case后面跟一个要比较的值和冒号。
- case语句中的值的数据类型必须与变量的数据类型相同，而且只能是常量或者字面常量。
- 当变量的值与case语句的值相等时，那么case语句之后的语句开始执行，直到break语句出现才会跳出switch语句。3
- 当遇到break语句时，switch语句终止。程序跳转到switch语句后面的语句执行。case语句不必须要包含break语句。如果没有break语句出现，程序会继续执行下一条case语句，直到出现break语句。
- switch语句可以包含一个default分支，该分支必须是switch语句的最后一个分支。default在没有case语句的值和变量值相等的时候执行。default分支不需要break语句。

## 实例

```
public class Test {

   public static void main(String args[]){
      //char grade = args[0].charAt(0);
      char grade = 'C';

      switch(grade)
      {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println("You passed");
         case 'F' :
            System.out.println("Better try again");
            break;
         default :
            System.out.println("Invalid grade");
      }
      System.out.println("Your grade is " + grade);
   }
}
```

以上代码编译运行结果如下：

```
$ java Test
Well done
Your grade is a C
$
```

# Java Number类

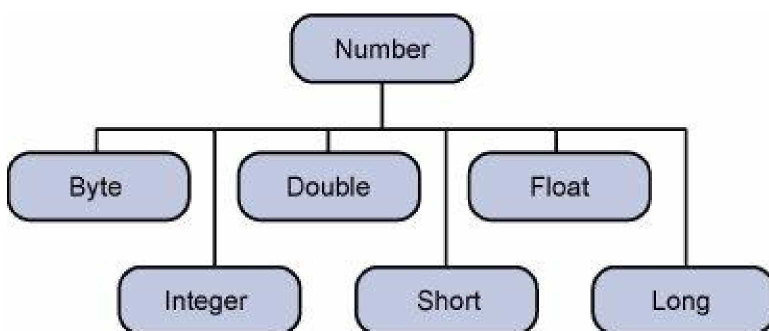
一般地，当需要使用数字的时候，我们通常使用内置数据类型，如：byte、int、long、double等。

## 实例

```
int i = 5000;
float gpa = 13.65;
byte mask = 0xaf;
```

然而，在实际开发过程中，我们经常会遇到需要使用对象，而不是内置数据类型的情形。为了解决这个问题，Java语言为每一个内置数据类型提供了对应的包装类。

所有的包装类（Integer、Long、Byte、Double、Float、Short）都是抽象类Number的子类。




这种由编译器特别支持的包装称为装箱，所以当内置数据类型被当作对象使用的时候，编译器会把内置类型装箱为包装类。相似的，编译器也可以把一个对象拆箱为内置类型。Number类属于java.lang包。

下面是一个装箱与拆箱的例子：

```
public class Test{

   public static void main(String args[]){
      Integer x = 5; // boxes int to an Integer object
      x =  x + 10;   // unboxes the Integer to a int
      System.out.println(x);
   }
}
```

以上实例编译运行结果如下：

```
15
```

当x被赋为整型值时，由于x是一个对象，所以编译器要对x进行装箱。然后，为了使x能进行加运算，所以要对x进行拆箱。

## Number类的成员方法

下面的表中列出的是Number类的方法：

| 方法 | 描述 |
| --- | --- |
| xxxValue() | 将number对象转换为xxx数据类型的值并返回。 |
| compareTo() | 将number对象与参数比较。 |
| equals() | 判断number对象是否与参数相等。 |
| valueOf() | 返回一个Integer对象指定的内置数据类型 |
| toString() | 以字符串形式返回值。 |
| parseInt() | 将字符串解析为int类型。 |
| abs() | 返回参数的绝对值。 |
| ceil() | 对整形变量向左取整，返回类型为double型。 |
| floor() | 对整型变量向右取整。返回类型为double类型。 |
| rint() | 返回与参数最接近的整数。返回类型为double。 |
| round() | 返回一个最接近的int、long型值。 |
| min() | 返回两个参数中的最小值。 |
| max() | 返回两个参数中的最大值。 |
| exp() | 返回自然数底数e的参数次方。 |
| log() | 返回参数的自然数底数的对数值。 |
| pow() | 返回第一个参数的第二个参数次方。 |
| sqrt() | 求参数的算术平方根。 |
| sin() | 求指定double类型参数的正弦值。 |
| cos() | 求指定double类型参数的余弦值。 |
| tan() | 求指定double类型参数的正切值。 |
| asin() | 求指定double类型参数的反正弦值。 |
| acos() | 求指定double类型参数的反余弦值。 |
| atan() | 求指定double类型参数的反正切值。 |
| atan2() | 将笛卡尔坐标转换为极坐标，并返回极坐标的角度值。 |
| toDegrees() | 将参数转化为角度。 |
| toRadians() | 将角度转换为弧度。 |
| random() | 返回一个随机数。 |

# Java Character类

使用字符时，我们通常使用的是内置数据类型char。

## 实例

```
char ch = 'a';

// Unicode for uppercase Greek omega character
char uniChar = '\u039A';

// 字符数组
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };
```

然而，在实际开发过程中，我们经常会遇到需要使用对象，而不是内置数据类型的情况。为了解决这个问题，Java语言为内置数据类型char提供了包装类Character类。

Character类提供了一系列方法来操纵字符。你可以使用Character的构造方法创建一个Character类对象，例如：

```
Character ch = new Character('a');
```

在某些情况下，Java编译器会自动创建一个Character对象。

例如，将一个char类型的参数传递给需要一个Character类型参数的方法时，那么编译器会自动地将char类型参数转换为Character对象。 这种特征称为装箱，反过来称为拆箱。

## 实例

```
// Here following primitive char 'a'
// is boxed into the Character object ch
Character ch = 'a';

// Here primitive 'x' is boxed for method test,
// return is unboxed to char 'c'
char c = test('x');
```

## 转义序列

前面有反斜杠（\）的字符代表转义字符，它对编译器来说是有特殊含义的。

下面列表展示了Java的转义序列：

| 转义序列 | 描述 |
| --- | --- |
| \t | 在文中该处插入一个tab键 |
| \b | 在文中该处插入一个后退键 |
| \n | 在文中该处换行 |
| \r | 在文中该处插入回车 |
| \f | 在文中该处插入换页符 |
| \' | 在文中该处插入单引号 |
| \" | 在文中该处插入双引号 |
| \ | 在文中该处插入反斜杠 |

## 实例

当打印语句遇到一个转义序列时，编译器可以正确地对其进行解释。

```
public class Test {

   public static void main(String args[]) {
      System.out.println("She said \"Hello!\" to me.");
   }
}
```

以上实例编译运行结果如下：

```
She said "Hello!" to me.
```

# Character 方法

下面是Character类的方法：

| 方法 | 描述 |
| --- | --- |
| isLetter() | 是否是一个字母 |
| isDigit() | 是否是一个数字字符 |
| isWhitespace() | 是否一个空格 |
| isUpperCase() | 是否是大写字母 |
| isLowerCase() | 是否是小写字母 |
| toUpperCase() | 指定字母的大写形式 |
| toLowerCase() | 指定字母的小写形式 |
| toString() | 返回字符的字符串形式，字符串的长度仅为1 |

对于方法的完整列表，请参考的java.lang.Character API规范。

# Java String类

字符串广泛应用在Java编程中，在Java中字符串属于对象，Java提供了String类来创建和操作字符串。

## 创建字符串

创建字符串最简单的方式如下:

```
String greeting = "Hello world!";
```

在代码中遇到字符串常量时，这里的值是"Hello world!"，编译器会使用该值创建一个String对象。

和其它对象一样，可以使用关键字和构造方法来创建String对象。

String类有11种构造方法，这些方法提供不同的参数来初始化字符串，比如提供一个字符数组参数:

```
public class StringDemo{

   public static void main(String args[]){
      char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};
      String helloString = new String(helloArray);
      System.out.println( helloString );
   }
}
```

以上实例编译运行结果如下：

```
hello.
```

注意:String类是不可改变的，所以你一旦创建了String对象，那它的值就无法改变了。 如果需要对字符串做很多修改，那么应该选择使用StringBuffer & StringBuilder 类。

## 字符串长度

用于获取有关对象的信息的方法称为访问器方法。

String类的一个访问器方法是length()方法，它返回字符串对象包含的字符数。

下面的代码执行后，len变量等于17:

```
public class StringDemo {

   public static void main(String args[]) {
      String palindrome = "Dot saw I was Tod";
      int len = palindrome.length();
      System.out.println( "String Length is : " + len );
   }
}
```

以上实例编译运行结果如下：

```
String Length is : 17
```

## 连接字符串

String类提供了连接两个字符串的方法：

```
string1.concat(string2);
```

返回string2连接string1的新字符串。也可以对字符串常量使用concat()方法，如：

```
"My name is ".concat("Zara");
```

更常用的是使用'+'操作符来连接字符串，如：

```
"Hello," + " world" + "!"
```

结果如下:

```
"Hello, world!"
```

下面是一个例子:

```
public class StringDemo {
   public static void main(String args[]) {
   String string1 = "saw I was ";
   System.out.println("Dot " + string1 + "Tod");
}
}
```

以上实例编译运行结果如下：

```
Dot saw I was Tod
```

# 创建格式化字符串

我们知道输出格式化数字可以使用printf()和format()方法。String类使用静态方法format()返回一个String对象而不是PrintStream对象。

String类的静态方法format()能用来创建可复用的格式化字符串，而不仅仅是用于一次打印输出。如下所示：

```
System.out.printf("The value of the float variable is " +
                  "%f, while the value of the integer " +
                  "variable is %d, and the string " +
                  "is %s", floatVar, intVar, stringVar);
```

你也可以这样写

```
String fs;
fs = String.format("The value of the float variable is " +
                   "%f, while the value of the integer " +
                   "variable is %d, and the string " +
                   "is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

# String 方法

下面是String类支持的方法，更多详细，参看Java API文档:

| 方法 | 描述 |
| --- | --- |
| char charAt(int index) | 返回指定索引处的 char 值。 |
| int compareTo(Object o) | 把这个字符串和另一个对象比较。 |
| int compareTo(String anotherString) | 按字典顺序比较两个字符串。 |

| | |
|---|---|
| int compareToIgnoreCase(String str) | 按字典顺序比较两个字符串，不考虑大小写。 |
| String concat(String str) | 将指定字符串连接到此字符串的结尾。 |
| boolean contentEquals(StringBuffer sb) | 当且仅当字符串与指定的StringButter有相同顺序的字符时候返回真。 |
| static String copyValueOf(char[] data) | 返回指定数组中表示该字符序列的 String。 |
| static String copyValueOf(char[] data, int offset, int count) | 返回指定数组中表示该字符序列的 String。 |
| boolean endsWith(String suffix) | 测试此字符串是否以指定的后缀结束。 |
| boolean equals(Object anObject) | 将此字符串与指定的对象比较。 |
| boolean equalsIgnoreCase(String anotherString) | 将此 String 与另一个 String 比较，不考虑大小写。 |
| byte[] getBytes() | 使用平台的默认字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。 |
| byte[] getBytes(String charsetName) | 使用指定的字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。 |
| void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) | 将字符从此字符串复制到目标字符数组。 |
| int hashCode() | 返回此字符串的哈希码。 |
| int indexOf(int ch) | 返回指定字符在此字符串中第一次出现处的索引。 |
| int indexOf(int ch, int fromIndex) | 返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索。 |
| int indexOf(String str) | 返回指定子字符串在此字符串中第一次出现处的索引。 |
| int indexOf(String str, int fromIndex) | 返回指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始。 |
| String intern() | 返回字符串对象的规范化表示形式。 |
| int lastIndexOf(int ch) | 返回指定字符在此字符串中最后一次出现处的索引。 |
| int lastIndexOf(int ch, int fromIndex) | 返回指定字符在此字符串中最后一次出现处的索引，从指定的索引处开始进行反向搜索。 |

| | |
|---|---|
| int lastIndexOf(String str) | 返回指定子字符串在此字符串中最右边出现处的索引。 |
| int lastIndexOf(String str, int fromIndex) | 返回指定子字符串在此字符串中最后一次出现处的索引，从指定的索引开始反向搜索。 |
| int length() | 返回此字符串的长度。 |
| boolean matches(String regex) | 告知此字符串是否匹配给定的正则表达式。 |
| boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len) | 测试两个字符串区域是否相等。 |
| boolean regionMatches(int toffset, String other, int ooffset, int len) | 测试两个字符串区域是否相等。 |
| String replace(char oldChar, char newChar) | 返回一个新的字符串，它是通过用 newChar 替换此字符串中出现的所有 oldChar 得到的。 |
| String replaceAll(String regex, String replacement) | 使用给定的 replacement 替换此字符串所有匹配给定的正则表达式的子字符串。 |
| String replaceFirst(String regex, String replacement) | 使用给定的 replacement 替换此字符串匹配给定的正则表达式的第一个子字符串。 |
| String[] split(String regex) | 根据给定正则表达式的匹配拆分此字符串。 |
| String[] split(String regex, int limit) | 根据匹配给定的正则表达式来拆分此字符串。 |
| boolean startsWith(String prefix) | 测试此字符串是否以指定的前缀开始。 |
| boolean startsWith(String prefix, int toffset) | 测试此字符串从指定索引开始的子字符串是否以指定前缀开始。 |
| CharSequence subSequence(int beginIndex, int endIndex) | 返回一个新的字符序列，它是此序列的一个子序列。 |
| String substring(int beginIndex) | 返回一个新的字符串，它是此字符串的一个子字符串。 |
| String substring(int beginIndex, int endIndex) | 返回一个新字符串，它是此字符串的一个子字符串。 |
| char[] toCharArray() | 将此字符串转换为一个新的字符数组。 |
| String toLowerCase() | 使用默认语言环境的规则将此 String 中 |

| | |
|---|---|
| String toLowerCase() | 的所有字符都转换为小写。 |
| String toLowerCase(Locale locale) | 使用给定 Locale 的规则将此 String 中的所有字符都转换为小写。 |
| String toString() | 返回此对象本身（它已经是一个字符串！）。 |
| String toUpperCase() | 使用默认语言环境的规则将此 String 中的所有字符都转换为大写。 |
| String toUpperCase(Locale locale) | 使用给定 Locale 的规则将此 String 中的所有字符都转换为大写。 |
| String trim() | 返回字符串的副本，忽略前导空白和尾部空白。 |
| static String valueOf(primitive data type x) | 返回给定data type类型x参数的字符串表示形式。 |

# Java StringBuffer和StringBuilder类

当对字符串进行修改的时候，需要使用StringBuffer和StringBuilder类。

和String类不同的是，StringBuffer和StringBuilder类的对象能够被多次的修改，并且不产生新的未使用对象。

StringBuilder类在Java 5中被提出，它和StringBuffer之间的最大不同在于StringBuilder的方法不是线程安全的（不能同步访问）。

由于StringBuilder相较于StringBuffer有速度优势，所以多数情况下建议使用StringBuilder类。然而在应用程序要求线程安全的情况下，则必须使用StringBuffer类。

## 实例

```
public class Test{

    public static void main(String args[]){
       StringBuffer sBuffer = new StringBuffer(" test");
       sBuffer.append(" String Buffer");
       System.ou.println(sBuffer);
   }
}
```

以上实例编译运行结果如下：

```
test String Buffer
```

# StringBuffer 方法

以下是StringBuffer类支持的主要方法：

| 方法 | 描述 |
| --- | --- |
| public StringBuffer append(String s) | 将指定的字符串追加到此字符序列。 |
| public StringBuffer reverse() | 将此字符序列用其反转形式取代。 |
| public delete(int start, int end) | 移除此序列的子字符串中的字符。 |
| public insert(int offset, int i) | 将 `int` 参数的字符串表示形式插入此序列中。 |
| replace(int start, int end, String str) | 使用给定 `String` 中的字符替换此序列的子字符串中的字符。 |

下面的列表里的方法和String类的方法类似：

| 方法 | 描述 |
| --- | --- |
| int capacity() | 返回当前容量。 |
| char charAt(int index) | 返回此序列中指定索引处的 `char` 值。 |
| void ensureCapacity(int minimumCapacity) | 确保容量至少等于指定的最小值。 |
| void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) | 将字符从此序列复制到目标字符数组 `dst` 。 |
| int indexOf(String str) | 返回第一次出现的指定子字符串在该字符串中的索引。 |
| int indexOf(String str, int fromIndex) | 从指定的索引处开始，返回第一次出现的指定子字符串在该字符串中的索引。 |
| int lastIndexOf(String str) | 返回最右边出现的指定子字符串在此字符串中的索引。 |
| int lastIndexOf(String str, int fromIndex) | 返回最后一次出现的指定子字符串在此字符串中的索引。 |
| int length() | 返回长度（字符数）。 |
| void setCharAt(int index, char ch) | 将给定索引处的字符设置为 `ch` 。 |
| void setLength(int newLength) | 设置字符序列的长度。 |
| CharSequence subSequence(int start, int end) | 返回一个新的字符序列，该字符序列是此序列的子序列。 |
| String substring(int start) | 返回一个新的 `String`，它包含此字符序列当前所包含的字符子序列。 |
| String substring(int start, int end) | 返回一个新的 `String` ，它包含此序列当前所包含的字符子序列。 |
| String toString() | 返回此序列中数据的字符串表示形式。 |

# Java 数组

数组对于每一门编辑应语言来说都是重要的数据结构之一，当然不同语言对数组的实现及处理也不尽相同。

Java语言中提供的数组是用来存储固定大小的同类型元素。

你可以声明一个数组变量，如numbers[100]来代替直接声明100个独立变量number0，number1，....，number99。

本教程将为大家介绍Java数组的声明、创建和初始化，并给出其对应的代码。

## 声明数组变量

首先必须声明数组变量，才能在程序中使用数组。下面是声明数组变量的语法：

```
dataType[] arrayRefVar;   // 首选的方法

或

dataType arrayRefVar[];  // 效果相同，但不是首选方法
```

注意: 建议使用dataType[] arrayRefVar 的声明风格声明数组变量。 dataType arrayRefVar[] 风格是来自 C/C++ 语言 ，在Java中采用是为了让 C/C++ 程序员能够快速理解java语言。

### 实例

下面是这两种语法的代码示例：

```
double[] myList;         // 首选的方法

或

double myList[];         //  效果相同，但不是首选方法
```

## 创建数组

Java语言使用new操作符来创建数组，语法如下：

```
arrayRefVar = new dataType[arraySize];
```

上面的语法语句做了两件事：

- 一、使用dataType[arraySize]创建了一个数组。
- 二、把新创建的数组的引用赋值给变量 arrayRefVar。

数组变量的声明，和创建数组可以用一条语句完成，如下所示：

```
dataType[] arrayRefVar = new dataType[arraySize];
```

另外，你还可以使用如下的方式创建数组。

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

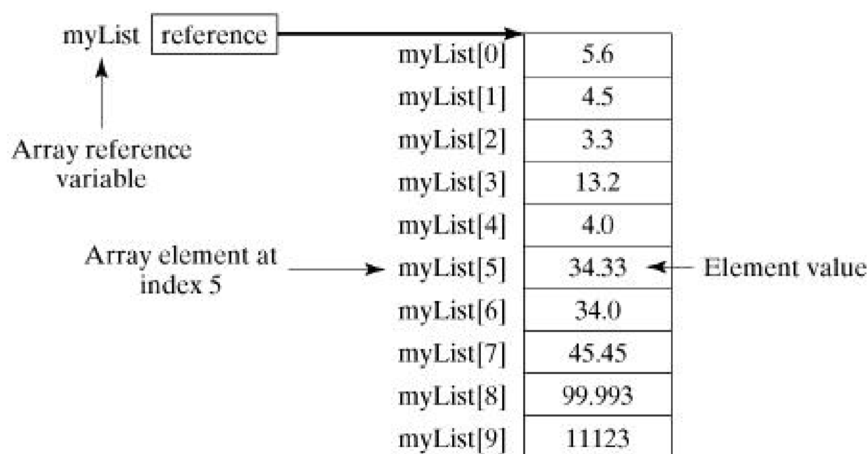
数组的元素是通过索引访问的。数组索引从0开始，所以索引值从0到arrayRefVar.length-1。

## 实例

下面的语句首先声明了一个数组变量myList，接着创建了一个包含10个double类型元素的数组，并且把它的引用赋值给myList变量。

```
double[] myList = new double[10];
```

下面的图片描绘了数组myList。这里myList数组里有10个double元素，它的下标从0到9。




# 处理数组

数组的元素类型和数组的大小都是确定的，所以当处理数组元素时候，我们通常使用基本循环或者foreach循环。

## 示例

该实例完整地展示了如何创建、初始化和操纵数组：

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // 打印所有数组元素
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }
      // 计算所有元素的总和
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      // 查找最大元素
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
         if (myList[i] > max) max = myList[i];
      }
      System.out.println("Max is " + max);
   }
}
```

以上实例编译运行结果如下：

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

# foreach循环

JDK 1.5 引进了一种新的循环类型，被称为foreach循环或者加强型循环，它能在不使用下标的情况下遍历数组。

## 示例

该实例用来显示数组myList中的所有元素：

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // 打印所有数组元素
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

以上实例编译运行结果如下：

```
1.9
2.9
3.4
3.5
```

## 数组作为函数的参数

数组可以作为参数传递给方法。例如，下面的例子就是一个打印int数组中元素的方法。

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

下面例子调用printArray方法打印出 3，1，2，6，4和2：

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

## 数组作为函数的返回值

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--)
    result[j] = list[i];
  }
  return result;
}
```

以上实例中result数组作为函数的返回值。

## Arrays 类

java.util.Arrays类能方便地操作数组，它提供的所有方法都是静态的。具有以下功能：

- 给数组赋值：通过fill方法。
- 对数组排序：通过sort方法,按升序。
- 比较数组：通过equals方法比较数组中元素值是否相等。
- 查找数组元素：通过binarySearch方法能对排序好的数组进行二分查找法操作。

具体说明请查看下表：

| 方法 | 说明 |
|---|---|
| **public static int binarySearch(Object[] a, Object key)** | 用二分查找算法在给定数组中搜索给定值的对象(Byte,Int,double等)。数组在调用前必须排序好的。如果查找值包含在数组中，则返回搜索键的索引；否则返回 (-(插入点) - 1)。 |
| **public static boolean equals(long[] a, long[] a2)** | 如果两个指定的 long 型数组彼此相等，则返回 true。如果两个数组包含相同数量的元素，并且两个数组中的所有相应元素对都是相等的，则认为这两个数组是相等的。换句话说，如果两个数组以相同顺序包含相同的元素，则两个数组是相等的。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。 |
| **public static void fill(int[] a, int val)** | 将指定的 int 值分配给指定 int 型数组指定范围中的每个元素。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。 |
| **public static void sort(Object[] a)** | 对指定对象数组根据其元素的自然顺序进行升序排列。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。 |

# Java 日期时间

java.util包提供了Date类来封装当前的日期和时间。 Date类提供两个构造函数来实例化Date对象。

第一个构造函数使用当前日期和时间来初始化对象。

```
Date( )
```

第二个构造函数接收一个参数，该参数是从1970年1月1日起的微秒数。

```
Date(long millisec)
```

Date对象创建以后，可以调用下面的方法。

| 方法 | 描述 |
|---|---|
| **boolean after(Date date)** | 若当调用此方法的Date对象在指定日期之后返回true,否则返回false。 |
| **boolean before(Date date)** | 若当调用此方法的Date对象在指定日期之前返回true,否则返回false。 |
| **Object clone( )** | 返回此对象的副本。 |
| **int compareTo(Date date)** | 比较当调用此方法的Date对象和指定日期。两者相等时候返回0。调用对象在指定日期之前则返回负数。调用对象在指定日期之后则返回正数。 |
| **int compareTo(Object obj)** | 若obj是Date类型则操作等同于compareTo(Date) 。否则它抛出ClassCastException。 |
| **boolean equals(Object date)** | 当调用此方法的Date对象和指定日期相等时候返回true,否则返回false。 |
| **long getTime( )** | 返回自 1970 年 1 月 1 日 00:00:00 GMT 以来此 Date 对象表示的毫秒数。 |
| **int hashCode( )** | 返回此对象的哈希码值。 |
| **void setTime(long time)** | 用自1970年1月1日00:00:00 GMT以后time毫秒数设置时间和日期。 |
| **String toString( )** | 转换Date对象为String表示形式，并返回该字符串。 |

# 获取当前日期时间

Java中获取当前日期和时间很简单，使用Date对象的 toString()方法来打印当前日期和时间，如下所示：

```
import java.util.Date;

public class DateDemo {
   public static void main(String args[]) {
       // 初始化 Date 对象
       Date date = new Date();

       // 使用 toString() 函数显示日期时间
       System.out.println(date.toString());
   }
}
```

以上实例编译运行结果如下:

```
Mon May 04 09:51:52 CDT 2013
```

# 日期比较

Java使用以下三种方法来比较两个日期：

- 使用getTime( ) 方法获取两个日期（自1970年1月1日经历的微妙数值），然后比较这两个值。
- 使用方法before()，after()和equals()。例如，一个月的12号比18号早，则new Date(99, 2, 12).before(new Date (99, 2, 18))返回true。
- 使用compareTo()方法，它是由Comparable接口定义的，Date类实现了这个接口。

# 使用**SimpleDateFormat**格式化日期

SimpleDateFormat是一个以语言环境敏感的方式来格式化和分析日期的类。SimpleDateFormat允许你选择任何用户自定义日期时间格式来运行。例如：

```
import java.util.*;
import java.text.*;

public class DateDemo {
   public static void main(String args[]) {

      Date dNow = new Date( );
      SimpleDateFormat ft =
      new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

      System.out.println("Current Date: " + ft.format(dNow));
   }
}
```

以上实例编译运行结果如下:

```
Current Date: Sun 2004.07.18 at 04:14:09 PM PDT
```

# 简单的DateFormat格式化编码

时间模式字符串用来指定时间格式。在此模式中，所有的ASCII字母被保留为模式字母，定义如下：

| 字母 | 描述 | 示例 |
|---|---|---|
| G | 纪元标记 | AD |
| y | 四位年份 | 2001 |
| M | 月份 | July or 07 |
| d | 一个月的日期 | 10 |
| h | A.M./P.M. (1~12)格式小时 | 12 |
| H | 一天中的小时 (0~23) | 22 |
| m | 分钟数 | 30 |
| s | 秒数 | 55 |
| S | 微妙数 | 234 |
| E | 星期几 | Tuesday |
| D | 一年中的日子 | 360 |
| F | 一个月中第几周的周几 | 2 (second Wed. in July) |
| w | 一年中第几周 | 40 |
| W | 一个月中第几周 | 1 |
| a | A.M./P.M. 标记 | PM |
| k | 一天中的小时(1~24) | 24 |
| K | A.M./P.M. (0~11)格式小时 | 10 |
| z | 时区 | Eastern Standard Time |
| ' | 文字定界符 | Delimiter |
| " | 单引号 | ` |

# 使用printf格式化日期

printf方法可以很轻松地格式化时间和日期。使用两个字母格式，它以t开头并且以下面表格中的一个字母结尾。例如：

```
import java.util.Date;

public class DateDemo {

  public static void main(String args[]) {
     // 初始化 Date 对象
     Date date = new Date();

     // 使用toString()显示日期和时间
     String str = String.format("Current Date/Time : %tc", date );

     System.out.printf(str);
  }
}
```

以上实例编译运行结果如下:

```
Current Date/Time : Sat Dec 15 16:37:57 MST 2012
```

如果你需要重复提供日期，那么利用这种方式来格式化它的每一部分就有点复杂了。因此，可以利用一个格式化字符串指出要被格式化的参数的索引。

索引必须紧跟在%后面，而且必须以$结束。例如：

```
import java.util.Date;

public class DateDemo {

   public static void main(String args[]) {
       // 初始化 Date 对象
       Date date = new Date();

       // 使用toString()显示日期和时间
       System.out.printf("%1$s %2$tB %2$td, %2$tY",
                         "Due date:", date);
   }
}
```

以上实例编译运行结果如下:

```
Due date: February 09, 2004
```

或者，你可以使用<标志。它表明先前被格式化的参数要被再次使用。例如：

```
import java.util.Date;

public class DateDemo {

   public static void main(String args[]) {
       // 初始化 Date 对象
       Date date = new Date();

       // 显示格式化时间
       System.out.printf("%s %tB %<te, %<tY",
                            "Due date:", date);
   }
}
```

以上实例编译运行结果如下:

```
Due date: February 09, 2004
```

## 日期和时间转换字符

| 字符 | 描述 | 例子 |
|---|---|---|
| c | 完整的日期和时间 | Mon May 04 09:51:52 CDT 2009 |
| F | ISO 8601 格式日期 | 2004-02-09 |
| D | U.S. 格式日期 (月/日/年) | 02/09/2004 |
| T | 24小时时间 | 18:05:19 |
| r | 12小时时间 | 06:05:19 pm |
| R | 24小时时间，不包含秒 | 18:05 |
| Y | 4位年份(包含前导0) | 2004 |
| y | 年份后2位(包含前导0) | 04 |
| C | 年份前2位(包含前导0) | 20 |
| B | 月份全称 | February |
| b | 月份简称 | Feb |
| n | 2位月份(包含前导0) | 02 |
| d | 2位日子(包含前导0) | 03 |
| | | |

| e | 2位日子(不包含前导0) | 9 |
|---|---|---|
| A | 星期全称 | Monday |
| a | 星期简称 | Mon |
| j | 3位年份(包含前导0) | 069 |
| H | 2位小时(包含前导0), 00 到 23 | 18 |
| k | 2位小时(不包含前导0), 0 到 23 | 18 |
| I | 2位小时(包含前导0), 01 到 12 | 06 |
| l | 2位小时(不包含前导0), 1 到 12 | 6 |
| M | 2位分钟(包含前导0) | 05 |
| S | 2位秒数(包含前导0) | 19 |
| L | 3位毫秒(包含前导0) | 047 |
| N | 9位纳秒(包含前导0) | 047000000 |
| P | 大写上下午标志 | PM |
| p | 小写上下午标志 | pm |
| z | 从GMT的RFC 822数字偏移 | -0800 |
| Z | 时区 | PST |
| s | 自 1970-01-01 00:00:00 GMT的秒数 | 1078884319 |
| Q | 自 1970-01-01 00:00:00 GMT的毫妙 | 1078884319047 |

还有其他有用的日期和时间相关的类。对于更多的细节，你可以参考到Java标准文档。

## 解析字符串为时间

SimpleDateFormat 类有一些附加的方法，特别是parse()，它试图按照给定的SimpleDateFormat 对象的格式化存储来解析字符串。例如：

```
import java.util.*;
import java.text.*;

public class DateDemo {

   public static void main(String args[]) {
      SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd");

      String input = args.length == 0 ? "1818-11-11" : args[0];

      System.out.print(input + " Parses as ");

      Date t;

      try {
          t = ft.parse(input);
          System.out.println(t);
      } catch (ParseException e) {
          System.out.println("Unparseable using " + ft);
      }
   }
}
```

以上实例编译运行结果如下:

```
$ java DateDemo
1818-11-11 Parses as Wed Nov 11 00:00:00 GMT 1818
$ java DateDemo 2007-12-01
2007-12-01 Parses as Sat Dec 01 00:00:00 GMT 2007
```

### Java 休眠(sleep)

你可以让程序休眠一毫秒的时间或者到您的计算机的寿命长的任意段时间。例如，下面的程序会休眠10秒：

```
import java.util.*;

public class SleepDemo {
   public static void main(String args[]) {
      try {
         System.out.println(new Date( ) + "\n");
         Thread.sleep(5*60*10);
         System.out.println(new Date( ) + "\n");
      } catch (Exception e) {
          System.out.println("Got an exception!");
      }
   }
}
```

以上实例编译运行结果如下:

```
Sun May 03 18:04:41 GMT 2009

Sun May 03 18:04:51 GMT 2009
```

# 测量时间

下面的一个例子表明如何测量时间间隔（以毫秒为单位）：

```
import java.util.*;

public class DiffDemo {

   public static void main(String args[]) {
      try {
         long start = System.currentTimeMillis( );
         System.out.println(new Date( ) + "\n");
         Thread.sleep(5*60*10);
         System.out.println(new Date( ) + "\n");
         long end = System.currentTimeMillis( );
         long diff = end - start;
         System.out.println("Difference is : " + diff);
      } catch (Exception e) {
         System.out.println("Got an exception!");
      }
   }
}
```

以上实例编译运行结果如下:

```
Sun May 03 18:16:51 GMT 2009

Sun May 03 18:16:57 GMT 2009

Difference is : 5993
```

# Calendar类

我们现在已经能够格式化并创建一个日期对象了，但是我们如何才能设置和获取日期数据的特定部分呢，比如说小时，日，或者分钟? 我们又如何在日期的这些部分加上或者减去值呢? 答案是使用Calendar 类。

Calendar类的功能要比Date类强大很多，而且在实现方式上也比Date类要复杂一些。

Calendar类是一个抽象类，在实际使用时实现特定的子类的对象，创建对象的过程对程序员来说是透明的，只需要使用getInstance方法创建即可。

## 创建一个代表系统当前日期的Calendar对象

```
Calendar c = Calendar.getInstance();//默认是当前日期
```

## 创建一个指定日期的Calendar对象

使用Calendar类代表特定的时间，需要首先创建一个Calendar的对象，然后再设定该对象中的年月日参数来完成。

```
//创建一个代表2009年6月12日的Calendar对象
Calendar c1 = Calendar.getInstance();
c1.set(2009, 6 - 1, 12);
```

## Calendar类对象字段类型

Calendar类中用一下这些常量表示不同的意义，jdk内的很多类其实都是采用的这种思想

| 常量 | 描述 |
| --- | --- |
| Calendar.YEAR | 年份 |
| Calendar.MONTH | 月份 |
| Calendar.DATE | 日期 |
| Calendar.DAY_OF_MONTH | 日期，和上面的字段意义完全相同 |
| Calendar.HOUR | 12小时制的小时 |
| Calendar.HOUR_OF_DAY | 24小时制的小时 |
| Calendar.MINUTE | 分钟 |
| Calendar.SECOND | 秒 |
| Calendar.DAY_OF_WEEK | 星期几 |

## Calendar类对象信息的设置

**Set设置**

如：

```
Calendar c1 = Calendar.getInstance();
```

调用：

```
public final void set(int year,int month,int date)
```

```
c1.set(2009, 6 - 1, 12);//把Calendar对象c1的年月日分别设这为：2009、6、1
```

利用字段类型设置

如果只设定某个字段，例如日期的值，则可以使用如下set方法：

```
public void set(int field,int value)
```

把 c1对象代表的日期设置为10号，其它所有的数值会被重新计算

```
c1.set(Calendar.DATE,10);
```

把c1对象代表的年份设置为2008年，其他的所有数值会被重新计算

```
c1.set(Calendar.YEAR,2008);
```

其他字段属性set的意义以此类推

**Add设置**

```
Calendar c1 = Calendar.getInstance();
```

把c1对象的日期加上10，也就是c1所表的日期的10天后的日期，其它所有的数值会被重新计算

```
c1.add(Calendar.DATE, 10);
```

把c1对象的日期加上10，也就是c1所表的日期的10天前的日期，其它所有的数值会被重新计算

```
<pre>c1.add(Calendar.DATE, -10);
```

其他字段属性的add的意义以此类推

## Calendar类对象信息的获得

```
Calendar c1 = Calendar.getInstance();
// 获得年份
int year = c1.get(Calendar.YEAR);
// 获得月份
int month = c1.get(Calendar.MONTH) + 1;
// 获得日期
int date = c1.get(Calendar.DATE);
// 获得小时
int hour = c1.get(Calendar.HOUR_OF_DAY);
// 获得分钟
int minute = c1.get(Calendar.MINUTE);
// 获得秒
int second = c1.get(Calendar.SECOND);
// 获得星期几（注意（这个与Date类是不同的）：1代表星期日、2代表星期1、3代表星
int day = c1.get(Calendar.DAY_OF_WEEK);
```

# GregorianCalendar类

Calendar类实现了公历日历，GregorianCalendar是Calendar类的一个具体实现。

Calendar 的getInstance（）方法返回一个默认用当前的语言环境和时区初始化的GregorianCalendar对象。GregorianCalendar定义了两个字段：AD和BC。这些代表公历定义的两个时代。

下面列出GregorianCalendar对象的几个构造方法：

| 构造函数 | 说明 |
| --- | --- |
| **GregorianCalendar()** | 在具有默认语言环境的默认时区内使用当前时间构造一个默认的 GregorianCalendar。 |
| **GregorianCalendar(int year, int month, int date)** | 在具有默认语言环境的默认时区内构造一个带有给定日期设置的 GregorianCalendar |
| **GregorianCalendar(int year, int month, int date, int hour, int minute)** | 为具有默认语言环境的默认时区构造一个具有给定日期和时间设置的 GregorianCalendar。 |
| **GregorianCalendar(int year, int month, int date, int hour, int minute, int second)** | 为具有默认语言环境的默认时区构造一个具有给定日期和时间设置的 GregorianCalendar。 |
| **GregorianCalendar(Locale aLocale)** | 在具有给定语言环境的默认时区内构造一个基于当前时间的 GregorianCalendar。 |
| **GregorianCalendar(TimeZone zone)** | 在具有默认语言环境的给定时区内构造一个基于当前时间的 GregorianCalendar。 |
| **GregorianCalendar(TimeZone zone, Locale aLocale)** | 在具有给定语言环境的给定时区内构造一个基于当前时间的 GregorianCalendar。 |

这里是GregorianCalendar 类提供的一些有用的方法列表：

| 方法 | 说明 |
| --- | --- |
| **void add(int field, int amount)** | 根据日历规则，将指定的（有符号的）时间量添加到给定的日历字段中。 |
| **protected void computeFields()** | 转换UTC毫秒值为时间域值 |
| **protected void computeTime()** | 覆盖Calendar ，转换时间域值为UTC毫秒值 |
| **boolean equals(Object obj)** | 比较此 GregorianCalendar 与指定的 Object。 |
| **int get(int field)** | 获取指定字段的时间值 |
| **int getActualMaximum(int field)** | 返回当前日期，给定字段的最大值 |
| **int getActualMinimum(int field)** | 返回当前日期，给定字段的最小值 |
| **int getGreatestMinimum(int field)** | 返回此 GregorianCalendar 实例给定日历字段的最高的最小值。 |

| | |
|---|---|
| **Date getGregorianChange()** | 获得格里高利历的更改日期。 |
| **int getLeastMaximum(int field)** | 返回此 GregorianCalendar 实例给定日历字段的最低的最大值 |
| **int getMaximum(int field)** | 返回此 GregorianCalendar 实例的给定日历字段的最大值。 |
| **Date getTime()** | 获取日历当前时间。 |
| **long getTimeInMillis()** | 获取用长整型表示的日历的当前时间 |
| **TimeZone getTimeZone()** | 获取时区。 |
| **int getMinimum(int field)** | 返回给定字段的最小值。 |
| **int hashCode()** | 重写hashCode. |
| **boolean isLeapYear(int year)** | 确定给定的年份是否为闰年。 |
| **void roll(int field, boolean up)** | 在给定的时间字段上添加或减去（上/下）单个时间单元，不更改更大的字段。 |
| **void set(int field, int value)** | 用给定的值设置时间字段。 |
| **void set(int year, int month, int date)** | 设置年、月、日的值。 |
| **void set(int year, int month, int date, int hour, int minute)** | 设置年、月、日、小时、分钟的值。 |
| **void set(int year, int month, int date, int hour, int minute, int second)** | 设置年、月、日、小时、分钟、秒的值。 |
| **void setGregorianChange(Date date)** | 设置 GregorianCalendar 的更改日期。 |
| **void setTime(Date date)** | 用给定的日期设置Calendar的当前时间。 |
| **void setTimeInMillis(long millis)** | 用给定的long型毫秒数设置Calendar的当前时间。 |
| **void setTimeZone(TimeZone value)** | 用给定时区值设置当前时区。 |
| **String toString()** | 返回代表日历的字符串。 |

## 实例

```
import java.util.*;

public class GregorianCalendarDemo {

   public static void main(String args[]) {
      String months[] = {
      "Jan", "Feb", "Mar", "Apr",
      "May", "Jun", "Jul", "Aug",
      "Sep", "Oct", "Nov", "Dec"};

      int year;
      // 初始化 Gregorian 日历
      // 使用当前时间和日期
      // 默认为本地时间和时区
      GregorianCalendar gcalendar = new GregorianCalendar();
      // 显示当前时间和日期的信息
      System.out.print("Date: ");
      System.out.print(months[gcalendar.get(Calendar.MONTH)]);
      System.out.print(" " + gcalendar.get(Calendar.DATE) + " ");
      System.out.println(year = gcalendar.get(Calendar.YEAR));
      System.out.print("Time: ");
      System.out.print(gcalendar.get(Calendar.HOUR) + ":");
      System.out.print(gcalendar.get(Calendar.MINUTE) + ":");
      System.out.println(gcalendar.get(Calendar.SECOND));

      // 测试当前年份是否为闰年
      if(gcalendar.isLeapYear(year)) {
         System.out.println("当前年份是闰年");
      }
      else {
         System.out.println("当前年份不是闰年");
      }
   }
}
```

以上实例编译运行结果如下：

```
Date: Apr 22 2009
Time: 11:25:27
当前年份不是闰年
```

关于Calender 类的完整列表，你可以参考标准的Java文档。

# Java正则表达式

正则表达式定义了字符串的模式。

正则表达式可以用来搜索、编辑或处理文本。

正则表达式并不仅限于某一种语言，但是在每种语言中有细微的差别。

Java正则表达式和Perl的是最为相似的。

java.util.regex包主要包括以下三个类：

- **Pattern**类：

  pattern对象是一个正则表达式的编译表示。Pattern类没有公共构造方法。要创建一个Pattern对象，你必须首先调用其公共静态编译方法，它返回一个Pattern对象。该方法接受一个正则表达式作为它的第一个参数。

- **Matcher**类：

  Matcher对象是对输入字符串进行解释和匹配操作的引擎。与Pattern类一样，Matcher也没有公共构造方法。你需要调用Pattern对象的matcher方法来获得一个Matcher对象。

- **PatternSyntaxException**：

  PatternSyntaxException是一个非强制异常类，它表示一个正则表达式模式中的语法错误。

## 捕获组

捕获组是把多个字符当一个单独单元进行处理的方法，它通过对括号内的字符分组来创建。

例如，正则表达式(dog) 创建了单一分组，组里包含"d"，"o"，和"g"。

捕获组是通过从左至右计算其开括号来编号。例如，在表达式（（A）（B（C））），有四个这样的组：

- ((A)(B(C)))
- (A)
- (B(C))
- (C)

可以通过调用matcher对象的groupCount方法来查看表达式有多少个分组。groupCount方法返回一个int值，表示matcher对象当前有多个捕获组。

还有一个特殊的组（组0），它总是代表整个表达式。该组不包括在groupCount的返回值中。

# 实例

下面的例子说明如何从一个给定的字符串中找到数字串：

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches
{
    public static void main( String args[] ){

      // 按指定模式在字符串查找
      String line = "This order was placed for QT3000! OK?";
      String pattern = "(.*)(\\d+)(.*)";

      // 创建 Pattern 对象
      Pattern r = Pattern.compile(pattern);

      // 现在创建 matcher 对象
      Matcher m = r.matcher(line);
      if (m.find( )) {
         System.out.println("Found value: " + m.group(0) );
         System.out.println("Found value: " + m.group(1) );
         System.out.println("Found value: " + m.group(2) );
      } else {
         System.out.println("NO MATCH");
      }
   }
}
```

以上实例编译运行结果如下：

```
Found value: This order was placed for QT3000! OK?
Found value: This order was placed for QT300
Found value: 0
```

# 正则表达式语法

| 字符 | 说明 |
|---|---|
| \ | 将下一字符标记为特殊字符、文本、反向引用或八进制转义符。例如，"n"匹配字符"n"。"\n"匹配换行符。序列"\\"匹配"\"，"\("匹配"("。 |
| ^ | 匹配输入字符串开始的位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，^ 还会与"\n"或"\r"之后的位置匹配。 |
| | |

| | |
|---|---|
| $ | 匹配输入字符串结尾的位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，$ 还会与"\n"或"\r"之前的位置匹配。 |
| * | 零次或多次匹配前面的字符或子表达式。例如，zo 匹配*"z"*和*"zoo"*。 等效于 {0,}。 |
| + | 一次或多次匹配前面的字符或子表达式。例如，"zo+"与"zo"和"zoo"匹配，但与"z"不匹配。+ 等效于 {1,}。 |
| ? | 零次或一次匹配前面的字符或子表达式。例如，"do(es)?"匹配"do"或"does"中的"do"。? 等效于 {0,1}。 |
| {*n*} | *n* 是非负整数。正好匹配 *n* 次。例如，"o{2}"与"Bob"中的"o"不匹配，但与"food"中的两个"o"匹配。 |
| {*n*,} | *n* 是非负整数。至少匹配 *n* 次。例如，"o{2,}"不匹配"Bob"中的"o"，而匹配"foooood"中的所有 o。"o{1,}"等效于"o+"。"o{0,}"等效于"o*"。 |
| {*n*,*m*} | *M* 和 *n* 是非负整数，其中 *n* <= *m*。匹配至少 *n* 次，至多 *m* 次。例如，"o{1,3}"匹配"fooooood"中的头三个 o。'o{0,1}' 等效于 'o?'。注意：您不能将空格插入逗号和数字之间。 |
| ? | 当此字符紧随任何其他限定符（*、+、?、{*n*}、{*n*,}、{*n*,*m*}）之后时，匹配模式是"非贪心的"。"非贪心的"模式匹配搜索到的、尽可能短的字符串，而默认的"贪心的"模式匹配搜索到的、尽可能长的字符串。例如，在字符串"oooo"中，"o+?"只匹配单个"o"，而"o+"匹配所有"o"。 |
| . | 匹配除"\n"之外的任何单个字符。若要匹配包括"\n"在内的任意字符，请使用诸如"[\s\S]"之类的模式。 |
| (*pattern*) | 匹配 *pattern* 并捕获该匹配的子表达式。可以使用 **$0…$9** 属性从结果"匹配"集合中检索捕获的匹配。若要匹配括号字符 ( )，请使用"("或者")"。 |
| (?:*pattern*) | 匹配 *pattern* 但不捕获该匹配的子表达式，即它是一个非捕获匹配，不存储供以后使用的匹配。这对于用"or"字符 (\|) 组合模式部件的情况很有用。例如，'industr(?:y\|ies) 是比 'industry\|industries' 更经济的表达式。 |
| (?=*pattern*) | 执行正向预测先行搜索的子表达式，该表达式匹配处于匹配 *pattern* 的字符串的起始点的字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?=95\|98\|NT\|2000)' 匹配"Windows 2000"中的"Windows"，但不匹配"Windows 3.1"中的"Windows"。预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。 |
| (?!*pattern*) | 执行反向预测先行搜索的子表达式，该表达式匹配不处于匹配 *pattern* 的字符串的起始点的搜索字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?!95\|98\|NT\|2000)' 匹配"Windows 3.1"中的 "Windows"，但不匹 |

| | |
|---|---|
| | 配"Windows 2000"中的"Windows"。预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。 |
| *x*\|*y* | 匹配 *x* 或 *y*。例如，'z\|food' 匹配"z"或"food"。'(z\|f)ood' 匹配"zood"或"food"。 |
| [*xyz*] | 字符集。匹配包含的任一字符。例如，"[abc]"匹配"plain"中的"a"。 |
| [^*xyz*] | 反向字符集。匹配未包含的任何字符。例如，"abc"匹配"plain"中的"p"。 |
| [*a-z*] | 字符范围。匹配指定范围内的任何字符。例如，"[a-z]"匹配"a"到"z"范围内的任何小写字母。 |
| [^*a-z*] | 反向范围字符。匹配不在指定的范围内的任何字符。例如，"[^a-z]"匹配任何不在"a"到"z"范围内的任何字符。 |
| \b | 匹配一个字边界，即字与空格间的位置。例如，"er\b"匹配"never"中的"er"，但不匹配"verb"中的"er"。 |
| \B | 非字边界匹配。"er\B"匹配"verb"中的"er"，但不匹配"never"中的"er"。 |
| \c*x* | 匹配 *x* 指示的控制字符。例如，\cM 匹配 Control-M 或回车符。*x* 的值必须在 A-Z 或 a-z 之间。如果不是这样，则假定 c 就是"c"字符本身。 |
| \d | 数字字符匹配。等效于 [0-9]。 |
| \D | 非数字字符匹配。等效于 0-9。 |
| \f | 换页符匹配。等效于 \x0c 和 \cL。 |
| \n | 换行符匹配。等效于 \x0a 和 \cJ。 |
| \r | 匹配一个回车符。等效于 \x0d 和 \cM。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等。与 [ \f\n\r\t\v] 等效。 |
| \S | 匹配任何非空白字符。与 [^ \f\n\r\t\v] 等效。 |
| \t | 制表符匹配。与 \x09 和 \cI 等效。 |
| \v | 垂直制表符匹配。与 \x0b 和 \cK 等效。 |
| \w | 匹配任何字类字符，包括下划线。与"[A-Za-z0-9_]"等效。 |
| \W | 与任何非单词字符匹配。与"[^A-Za-z0-9_]"等效。 |
| \x*n* | 匹配 *n*，此处的 *n* 是一个十六进制转义码。十六进制转义码必须正好是两位数长。例如，"\x41"匹配"A"。"\x041"与"\x04"&"1"等效。允许在正则表达式中使用 ASCII 代码。 |

| | |
|---|---|
| _num_ | 匹配 *num*，此处的 *num* 是一个正整数。到捕获匹配的反向引用。例如，"(.)\1"匹配两个连续的相同字符。 |
| _n_ | 标识一个八进制转义码或反向引用。如果 _n 前面至少有 _*n* 个捕获子表达式，那么 *n* 是反向引用。否则，如果 *n* 是八进制数 (0-7)，那么 *n* 是八进制转义码。 |
| _nm_ | 标识一个八进制转义码或反向引用。如果 _nm 前面至少有 _*nm* 个捕获子表达式，那么 *nm* 是反向引用。如果 _nm 前面至少有 _*n* 个捕获，则 *n* 是反向引用，后面跟有字符 *m*。如果两种前面的情况都不存在，则 _nm 匹配八进制值 _*nm*，其中 *n* 和 *m* 是八进制数字 (0-7)。 |
| \nml | 当 *n* 是八进制数 (0-3)，*m* 和 *l* 是八进制数 (0-7) 时，匹配八进制转义码 *nml*。 |
| \u*n* | 匹配 *n*，其中 *n* 是以四位十六进制数表示的 Unicode 字符。例如，\u00A9 匹配版权符号 (?)。 |

# Mather类的方法

## 索引方法

索引方法提供了有用的索引值，精确表明输入字符串中在哪能找到匹配：

| 方法 | 说明 |
|---|---|
| **public int start()** | 返回以前匹配的初始索引。 |
| **public int start(int group)** | 返回在以前的匹配操作期间，由给定组所捕获的子序列的初始索引 |
| **public int end()** | 返回最后匹配字符之后的偏移量。 |
| **public int end(int group)** | 返回在以前的匹配操作期间，由给定组所捕获子序列的最后字符之后的偏移量。 |

## 研究方法

研究方法用来检查输入字符串并返回一个布尔值，表示是否找到该模式：

| 方法 | 说明 | |
|---|---|---|
| **public boolean lookingAt()** | 尝试将从区域开头开始的输入序列与该模式匹配。 | |
| **public boolean find()** | 尝试查找与该模式匹配的输入序列的下一个子序列。 | |
| **public boolean find(int start** | ) | 重置此匹配器，然后尝试查找匹配该模式、从指定索引开始的输入序列的下一个子序列。 |
| **public boolean matches()** | 尝试将整个区域与模式匹配。 | |

# 替换方法

替换方法是替换输入字符串里文本的方法：

| 方法 | 说明 |
|---|---|
| **public Matcher appendReplacement(StringBuffer sb, String replacement)** | 实现非终端添加和替换步骤。 |
| **public StringBuffer appendTail(StringBuffer sb)** | 实现终端添加和替换步骤。 |
| **public String replaceAll(String replacement)** | 替换模式与给定替换字符串相匹配的输入序列的每个子序列。 |
| **public String replaceFirst(String replacement)** | 替换模式与给定替换字符串匹配的输入序列的第一个子序列。 |
| **public static String quoteReplacement(String s)** | 返回指定字符串的字面替换字符串。这个方法返回一个字符串，就像传递给Matcher类的appendReplacement 方法一个字面字符串一样工作。 |

# start 和end 方法

下面是一个对单词"cat"出现在输入字符串中出现次数进行计数的例子：

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches
{
    private static final String REGEX = "\\bcat\\b";
    private static final String INPUT =
                                    "cat cat cat cattie cat";

    public static void main( String args[] ){
       Pattern p = Pattern.compile(REGEX);
       Matcher m = p.matcher(INPUT); // 获取 matcher 对象
       int count = 0;

       while(m.find()) {
         count++;
         System.out.println("Match number "+count);
         System.out.println("start(): "+m.start());
         System.out.println("end(): "+m.end());
      }
   }
}
```

以上实例编译运行结果如下：

```
Match number 1
start(): 0
end(): 3
Match number 2
start(): 4
end(): 7
Match number 3
start(): 8
end(): 11
Match number 4
start(): 19
end(): 22
```

可以看到这个例子是使用单词边界，以确保字母 "c" "a" "t" 并非仅是一个较长的词的子串。它也提供了一些关于输入字符串中匹配发生位置的有用信息。

Start方法返回在以前的匹配操作期间，由给定组所捕获的子序列的初始索引，end方法最后一个匹配字符的索引加1。

## matches 和lookingAt 方法

matches 和lookingAt 方法都用来尝试匹配一个输入序列模式。它们的不同是matcher要求整个序列都匹配，而lookingAt 不要求。

这两个方法经常在输入字符串的开始使用。

我们通过下面这个例子，来解释这个功能：

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches
{
    private static final String REGEX = "foo";
    private static final String INPUT = "fooooooooooooooooo";
    private static Pattern pattern;
    private static Matcher matcher;

    public static void main( String args[] ){
       pattern = Pattern.compile(REGEX);
       matcher = pattern.matcher(INPUT);

       System.out.println("Current REGEX is: "+REGEX);
       System.out.println("Current INPUT is: "+INPUT);

       System.out.println("lookingAt(): "+matcher.lookingAt());
       System.out.println("matches(): "+matcher.matches());
   }
}
```

以上实例编译运行结果如下：

```
Current REGEX is: foo
Current INPUT is: fooooooooooooooooo
lookingAt(): true
matches(): false
```

# replaceFirst 和replaceAll 方法

replaceFirst 和replaceAll 方法用来替换匹配正则表达式的文本。不同的是，replaceFirst 替换首次匹配，replaceAll 替换所有匹配。

下面的例子来解释这个功能：

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches
{
    private static String REGEX = "dog";
    private static String INPUT = "The dog says meow. " +
                                    "All dogs say meow.";
    private static String REPLACE = "cat";

    public static void main(String[] args) {
       Pattern p = Pattern.compile(REGEX);
       // get a matcher object
       Matcher m = p.matcher(INPUT);
       INPUT = m.replaceAll(REPLACE);
       System.out.println(INPUT);
   }
}
```

以上实例编译运行结果如下：

```
The cat says meow. All cats say meow.
```

## appendReplacement 和 appendTail 方法

Matcher 类也提供了appendReplacement 和appendTail 方法用于文本替换：

看下面的例子来解释这个功能：

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches
{
   private static String REGEX = "a*b";
   private static String INPUT = "aabfooaabfooabfoob";
   private static String REPLACE = "-";
   public static void main(String[] args) {
      Pattern p = Pattern.compile(REGEX);
      // 获取 matcher 对象
      Matcher m = p.matcher(INPUT);
      StringBuffer sb = new StringBuffer();
      while(m.find()){
         m.appendReplacement(sb,REPLACE);
      }
      m.appendTail(sb);
      System.out.println(sb.toString());
   }
}
```

以上实例编译运行结果如下：

```
-foo-foo-foo-
```

## PatternSyntaxException 类的方法

PatternSyntaxException 是一个非强制异常类，它指示一个正则表达式模式中的语法错误。

PatternSyntaxException 类提供了下面的方法来帮助我们查看发生了什么错误。

| 方法 | 说明 |
| --- | --- |
| **public String getDescription()** | 获取错误的描述。 |
| **public int getIndex()** | 获取错误的索引。 |
| **public String getPattern()** | 获取错误的正则表达式模式。 |
| **public String getMessage()** | 返回多行字符串，包含语法错误及其索引的描述、错误的正则表达式模式和模式中错误索引的可视化指示。 |

# Java 方法

在前面几个章节中我们经常使用到System.out.println()，那么它是什么呢？

println()是一个方法(Method)，而System是系统类(Class)，out是标准输出对象(Object)。这句话的用法是调用系统类System中的标准输出对象out中的方法println()。

## 那么什么是方法呢？

Java方法是语句的集合，它们在一起执行一个功能。

- 方法是解决一类问题的步骤的有序组合
- 方法包含于类或对象中
- 方法在程序中被创建，在其他地方被引用

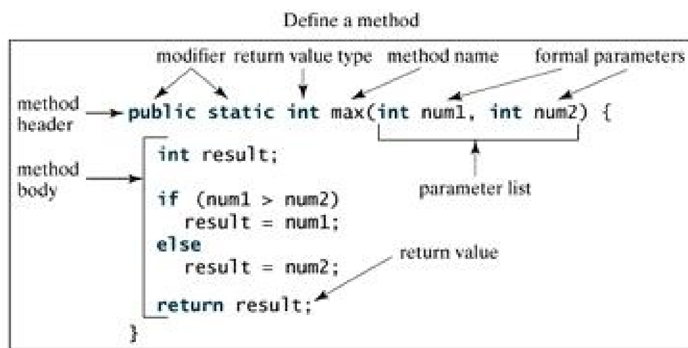
# 方法的定义

一般情况下，定义一个方法包含以下语法：

```
修饰符 返回值类型 方法名 (参数类型 参数名){
    ...
    方法体
    ...
    return 返回值;
}
```

方法包含一个方法头和一个方法体。下面是一个方法的所有部分：

- 修饰符：修饰符，这是可选的，告诉编译器如何调用该方法。定义了该方法的访问类型。
- 返回值类型 ：方法可能会返回值。returnValueType是方法返回值的数据类型。有些方法执行所需的操作，但没有返回值。在这种情况下，returnValueType是关键字**void**。
- 方法名：是方法的实际名称。方法名和参数表共同构成方法签名。
- 参数类型：参数像是一个占位符。当方法被调用时，传递值给参数。这个值被称为实参或变量。参数列表是指方法的参数类型、顺序和参数的个数。参数是可选的，方法可以不包含任何参数。
- 方法体：方法体包含具体的语句，定义该方法的功能。




如：

```
public static int age(int birthday){...}
```

参数可以有多个：

```
static float interest(float principal, int year){...}
```

注意：在一些其它语言中方法指过程和函数。一个返回非void类型返回值的方法称为函数；一个返回void类型返回值的方法叫做过程。

## 实例

下面的方法包含2个参数num1和num2，它返回这两个参数的最大值。

```
/** 返回两个整型变量数据的较大值 */
public static int max(int num1, int num2) {
   int result;
   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

# 方法调用

Java支持两种调用方法的方式，根据方法是否返回值来选择。

当程序调用一个方法时，程序的控制权交给了被调用的方法。当被调用方法的返回语句执行或者到达方法体闭括号时候交还控制权给程序。

当方法返回一个值的时候，方法调用通常被当做一个值。例如：

```
int larger = max(30, 40);
```

如果方法返回值是void，方法调用一定是一条语句。例如，方法println返回void。下面的调用是个语句：

```
System.out.println("Welcome to Java!");
```

## 示例

下面的例子演示了如何定义一个方法，以及如何调用它：

```
public class TestMax {
   /** 主方法 */
   public static void main(String[] args) {
      int i = 5;
      int j = 2;
      int k = max(i, j);
      System.out.println("The maximum between " + i +
                    " and " + j + " is " + k);
   }

   /** 返回两个整数变量较大的值 */
   public static int max(int num1, int num2) {
      int result;
      if (num1 > num2)
         result = num1;
      else
         result = num2;

      return result;
   }
}
```

以上实例编译运行结果如下：

```
The maximum between 5 and 2 is 5
```

这个程序包含main方法和max方法。Main方法是被JVM调用的，除此之外，main方法和其它方法没什么区别。

main方法的头部是不变的，如例子所示，带修饰符public和static,返回void类型值，方法名字是main,此外带个一个String[]类型参数。String[]表明参数是字符串数组。

# void 关键字

本节说明如何声明和调用一个void方法。

下面的例子声明了一个名为printGrade的方法，并且调用它来打印给定的分数。

## 示例

```
public class TestVoidMethod {

   public static void main(String[] args) {
      printGrade(78.5);
   }

   public static void printGrade(double score) {
      if (score >= 90.0) {
         System.out.println('A');
      }
      else if (score >= 80.0) {
         System.out.println('B');
      }
      else if (score >= 70.0) {
         System.out.println('C');
      }
      else if (score >= 60.0) {
         System.out.println('D');
      }
      else {
         System.out.println('F');
      }
   }
}
```

以上实例编译运行结果如下：

```
C
```

这里printGrade方法是一个void类型方法，它不返回值。

一个void方法的调用一定是一个语句。 所以，它被在main方法第三行以语句形式调用。就像任何以分号结束的语句一样。

# 通过值传递参数

调用一个方法时候需要提供参数，你必须按照参数列表指定的顺序提供。

例如，下面的方法连续n次打印一个消息：

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

## 示例

下面的例子演示按值传递的效果。

该程序创建一个方法，该方法用于交换两个变量。

```
public class TestPassByValue {

   public static void main(String[] args) {
      int num1 = 1;
      int num2 = 2;

      System.out.println("Before swap method, num1 is " +
                          num1 + " and num2 is " + num2);

      // 调用swap方法
      swap(num1, num2);
      System.out.println("After swap method, num1 is " +
                         num1 + " and num2 is " + num2);
   }
   /** 交换两个变量的方法 */
   public static void swap(int n1, int n2) {
      System.out.println("\tInside the swap method");
      System.out.println("\t\tBefore swapping n1 is " + n1
                           + " n2 is " + n2);
      // 交换 n1 与 n2的值
      int temp = n1;
      n1 = n2;
      n2 = temp;

      System.out.println("\t\tAfter swapping n1 is " + n1
                           + " n2 is " + n2);
   }
}
```

以上实例编译运行结果如下：

```
Before swap method, num1 is 1 and num2 is 2
        Inside the swap method
                Before swapping n1 is 1 n2 is 2
                After swapping n1 is 2 n2 is 1
After swap method, num1 is 1 and num2 is 2
```

传递两个参数调用swap方法。有趣的是，方法被调用后，实参的值并没有改变。

# 方法的重载

上面使用的max方法仅仅适用于int型数据。但如果你想得到两个浮点类型数据的最大值呢？

解决方法是创建另一个有相同名字但参数不同的方法，如下面代码所示：

```
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```

如果你调用max方法时传递的是int型参数，则 int型参数的max方法就会被调用；

如果传递的事double型参数，则double类型的max方法体会被调用，这叫做方法重载；

就是说一个类的两个方法拥有相同的名字，但是有不同的参数列表。

Java编译器根据方法签名判断哪个方法应该被调用。

方法重载可以让程序更清晰易读。执行密切相关任务的方法应该使用相同的名字。

重载的方法必须拥有不同的参数列表。你不能仅仅依据修饰符或者返回类型的不同来重载方法。

# 变量作用域

变量的范围是程序中该变量可以被引用的部分。

方法内定义的变量被称为局部变量。

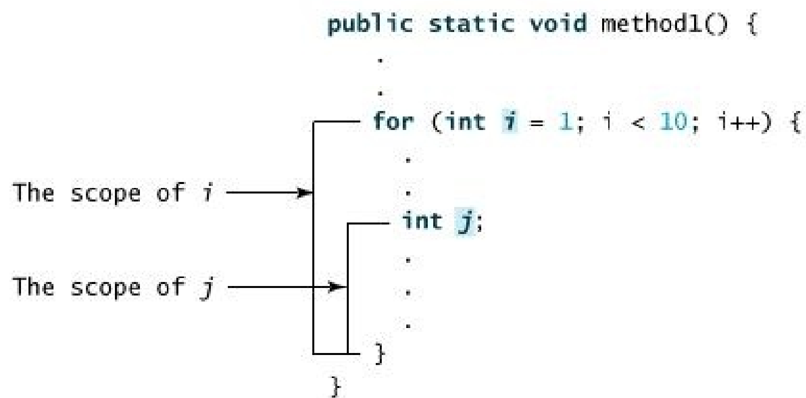
局部变量的作用范围从声明开始，直到包含它的块结束。

局部变量必须声明才可以使用。

方法的参数范围涵盖整个方法。参数实际上是一个局部变量。

for循环的初始化部分声明的变量，其作用范围在整个循环。

但循环体内声明的变量其适用范围是从它声明到循环体结束。它包含如下所示的变量声明：




你可以在一个方法里，不同的非嵌套块中多次声明一个具有相同的名称局部变量，但你不能在嵌套块内两次声明局部变量。

# 命令行参数的使用

有时候你希望运行一个程序时候再传递给它消息。这要靠传递命令行参数给main()函数实现。

命令行参数是在执行程序时候紧跟在程序名字后面的信息。

## 实例

下面的程序打印所有的命令行参数：

```
public class CommandLine {

   public static void main(String args[]){
      for(int i=0; i<args.length; i++){
         System.out.println("args[" + i + "]: " +
                                           args[i]);
      }
   }
}
```

如下所示，运行这个程序：

```
java CommandLine this is a command line 200 -100
```

运行结果如下：

```
args[0]: this
args[1]: is
args[2]: a
args[3]: command
args[4]: line
args[5]: 200
args[6]: -100
```

# 构造方法

当一个对象被创建时候，构造方法用来初始化该对象。构造方法和它所在类的名字相同，但构造方法没有返回值。

通常会使用构造方法给一个类的实例变量赋初值，或者执行其它必要的步骤来创建一个完整的对象。

不管你与否自定义构造方法，所有的类都有构造方法，因为Java自动提供了一个默认构造方法，它把所有成员初始化为0。

一旦你定义了自己的构造方法，默认构造方法就会失效。

## 实例

下面是一个使用构造方法的例子：

```
// 一个简单的构造函数
class MyClass {
   int x;

   // 以下是构造函数
   MyClass() {
      x = 10;
   }
}
```

你可以像下面这样调用构造方法来初始化一个对象：

```
public class ConsDemo {

   public static void main(String args[]) {
      MyClass t1 = new MyClass();
      MyClass t2 = new MyClass();
      System.out.println(t1.x + " " + t2.x);
   }
}
```

大多时候需要一个有参数的构造方法。

### 实例

下面是一个使用构造方法的例子：

```
// 一个简单的构造函数
class MyClass {
   int x;

   // 以下是构造函数
   MyClass(int i ) {
      x = i;
   }
}
```

你可以像下面这样调用构造方法来初始化一个对象：

```
public class ConsDemo {

   public static void main(String args[]) {
      MyClass t1 = new MyClass( 10 );
      MyClass t2 = new MyClass( 20 );
      System.out.println(t1.x + " " + t2.x);
   }
}
```

运行结果如下：

```
10 20
```

## 可变参数

JDK 1.5 开始，Java支持传递同类型的可变参数给一个方法。

方法的可变参数的声明如下所示：

```
typeName... parameterName
```

在方法声明中，在指定参数类型后加一个省略号(...) 。

一个方法中只能指定一个可变参数，它必须是方法的最后一个参数。任何普通的参数必须在它之前声明。

## 实例

```
public class VarargsDemo {

   public static void main(String args[]) {
      // 调用可变参数的方法
      printMax(34, 3, 3, 2, 56.5);
      printMax(new double[]{1, 2, 3});
   }

   public static void printMax( double... numbers) {
   if (numbers.length == 0) {
      System.out.println("No argument passed");
      return;
   }

   double result = numbers[0];

   for (int i = 1; i <  numbers.length; i++)
      if (numbers[i] >  result)
      result = numbers[i];
      System.out.println("The max value is " + result);
   }
}
```

以上实例编译运行结果如下：

```
The max value is 56.5
The max value is 3.0
```

# finalize() 方法

Java允许定义这样的方法，它在对象被垃圾收集器析构(回收)之前调用，这个方法叫做finalize( )，它用来清除回收对象。

例如，你可以使用finalize()来确保一个对象打开的文件被关闭了。

在finalize()方法里，你必须指定在对象销毁时候要执行的操作。

finalize()一般格式是：

```
protected void finalize()
{
   // 在这里终结代码
}
```

关键字protected是一个限定符，它确保finalize() 方法不会被该类以外的代码调用。

当然，Java的内存回收可以由JVM来自动完成。如果你手动使用，则可以使用上面的方法。

## 实例

```
public class FinalizationDemo {
    public static void main(String[] args) {
        Cake c1 = new Cake(1);
        Cake c2 = new Cake(2);
        Cake c3 = new Cake(3);

        c2 = c3 = null;
        System.gc(); //调用Java垃圾收集器
    }
}

class Cake extends Object {
    private int id;
    public Cake(int id) {
        this.id = id;
        System.out.println("Cake Object " + id + "is created");
    }

    protected void finalize() throws java.lang.Throwable {
        super.finalize();
        System.out.println("Cake Object " + id + "is disposed");
    }
}
```

运行以上代码，输出结果如下：

```
C:\1>java FinalizationDemo
Cake Object 1is created
Cake Object 2is created
Cake Object 3is created
Cake Object 3is disposed
Cake Object 2is disposed
```

# Java 流(Stream)、文件(File)和IO

Java.io包几乎包含了所有操作输入、输出需要的类。所有这些流类代表了输入源和输出目标。

Java.io包中的流支持很多种格式，比如：基本类型、对象、本地化字符集等等。

一个流可以理解为一个数据的序列。输入流表示从一个源读取数据，输出流表示向一个目标写数据。

Java为I/O提供了强大的而灵活的支持，使其更广泛地应用到文件传输和网络编程中。

但本节讲述最基本的和流与I/O相关的功能。我们将通过一个个例子来学习这些功能。

## 读取控制台输入

Java的控制台输入由Sysem.in完成。

为了获得一个绑定到控制台的字符流，你可以把System.in包装在一个BufferedReader 对象中来创建一个字符流。

下面是创建BufferedReader的基本语法：

```
BufferedReader br = new BufferedReader(new
                        InputStreamReader(System.in));
```

BufferedReader对象创建后，我们便可以使用read()方法从控制台读取一个字符，或者用readLine()方法读取一个字符串。

## 从控制台读取多字符输入

从BufferedReader对象读取一个字符要使用read()方法，它的语法如下：

```
int read( ) throws IOException
```

每次调用read()方法，它从输入流读取一个字符并把该字符作为整数值返回。 当流结束的时候返回-1。该方法抛出IOException。

下面的程序示范了用read()方法从控制台不断读取字符直到用户输入"q"。

```
// 使用 BufferedReader 在控制台读取字符

import java.io.*;

public class BRRead {
   public static void main(String args[]) throws IOException
   {
      char c;
      // 使用 System.in 创建 BufferedReader
      BufferedReader br = new BufferedReader(new
                         InputStreamReader(System.in));
      System.out.println("Enter characters, 'q' to quit.");
      // 读取字符
      do {
         c = (char) br.read();
         System.out.println(c);
      } while(c != 'q');
   }
}
```

以上实例编译运行结果如下:

```
Enter characters, 'q' to quit.
123abcq
1
2
3
a
b
c
q
```

# 从控制台读取字符串

从标准输入读取一个字符串需要使用BufferedReader的readLine()方法。

它的一般格式是：

```
String readLine( ) throws IOException
```

下面的程序读取和显示字符行直到你输入了单词"end"。

```
// 使用 BufferedReader 在控制台读取字符
import java.io.*;
public class BRReadLines {
   public static void main(String args[]) throws IOException
   {
      // 使用 System.in 创建 BufferedReader
      BufferedReader br = new BufferedReader(new
                              InputStreamReader(System.in));
      String str;
      System.out.println("Enter lines of text.");
      System.out.println("Enter 'end' to quit.");
      do {
         str = br.readLine();
         System.out.println(str);
      } while(!str.equals("end"));
   }
}
```

以上实例编译运行结果如下:

```
Enter lines of text.
Enter 'end' to quit.
This is line one
This is line one
This is line two
This is line two
end
end
```

# 控制台输出

在此前已经介绍过，控制台的输出由 print( ) 和println( )完成。这些方法都由类 PrintStream 定义，System.out是该类对象的一个引用。

PrintStream 继承了OutputStream类，并且实现了方法write()。这样，write()也可以用来往控制台写操作。

PrintStream 定义write()的最简单格式如下所示：

```
void write(int byteval)
```

该方法将byteval的低八位字节写到流中。

## 实例

下面的例子用write()把字符"A"和紧跟着的换行符输出到屏幕：

```
import java.io.*;

// 演示 System.out.write().
public class WriteDemo {
   public static void main(String args[]) {
      int b;
      b = 'A';
      System.out.write(b);
      System.out.write('\n');
   }
}
```

运行以上实例在输出窗口输出"A"字符

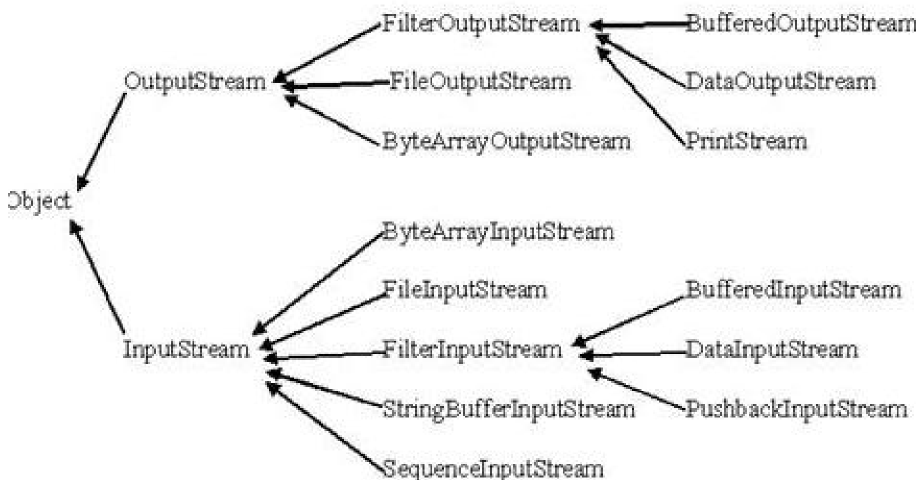
```
A
```

注意：write()方法不经常使用，因为print()和println()方法用起来更为方便。

# 读写文件

如前所述，一个流被定义为一个数据序列。输入流用于从源读取数据，输出流用于向目标写数据。

下图是一个描述输入流和输出流的类层次图。




下面将要讨论的两个重要的流是FileInputStream 和FileOutputStream：

# FileInputStream

该流用于从文件读取数据，它的对象可以用关键字new来创建。

有多种构造方法可用来创建对象。

可以使用字符串类型的文件名来创建一个输入流对象来读取文件：

```
InputStream f = new FileInputStream("C:/java/hello");
```

也可以使用一个文件对象来创建一个输入流对象来读取文件。我们首先得使用File()方法来创建一个文件对象：

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

创建了InputStream对象，就可以使用下面的方法来读取流或者进行其他的流操作。

| 方法 | 描述 |
|---|---|
| **public void close() throws IOException{}** | 关闭此文件输入流并释放与此流有关的所有系统资源。抛出IOException异常。 |
| **protected void finalize()throws IOException {}** | 这个方法清除与该文件的连接。确保在不再引用文件输入流时调用其 close 方法。抛出IOException异常。 |
| **public int read(int r)throws IOException{}** | 这个方法从InputStream对象读取指定字节的数据。返回为整数值。返回下一字节数据，如果已经到结尾则返回-1。 |
| **public int read(byte[] r) throws IOException{}** | 这个方法从输入流读取r.length长度的字节。返回读取的字节数。如果是文件结尾则返回-1。 |
| **public int available() throws IOException{}** | 返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取的字节数。返回一个整数值。 |

除了InputStream外，还有一些其他的输入流，更多的细节参考下面链接：

- ByteArrayInputStream
- DataInputStream

# FileOutputStream

该类用来创建一个文件并向文件中写数据。

如果该流在打开文件进行输出前，目标文件不存在，那么该流会创建该文件。

有两个构造方法可以用来创建FileOutputStream 对象。

使用字符串类型的文件名来创建一个输出流对象：

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

也可以使用一个文件对象来创建一个输出流来写文件。我们首先得使用File()方法来创建一个文件对象：

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

创建OutputStream 对象完成后，就可以使用下面的方法来写入流或者进行其他的流操作。

| 序号 | 方法及描述 |
|---|---|
| **public void close() throws IOException{}** | 关闭此文件输入流并释放与此流有关的所有系统资源。抛出IOException异常。 |
| **protected void finalize()throws IOException {}** | 这个方法清除与该文件的连接。确保在不再引用文件输入流时调用其 close 方法。抛出IOException异常。 |
| **public void write(int w)throws IOException{}** | 这个方法把指定的字节写到输出流中。 |
| **public void write(byte[] w)** | 把指定数组中w.length长度的字节写到OutputStream中。 |

除了OutputStream外，还有一些其他的输出流，更多的细节参考下面链接：

- ByteArrayOutputStream
- DataOutputStream

## 实例

下面是一个演示InputStream和OutputStream用法的例子：

```
import java.io.*;

public class fileStreamTest{

   public static void main(String args[]){

   try{
      byte bWrite [] = {11,21,3,40,5};
      OutputStream os = new FileOutputStream("test.txt");
      for(int x=0; x < bWrite.length ; x++){
         os.write( bWrite[x] ); // writes the bytes
      }
      os.close();

      InputStream is = new FileInputStream("test.txt");
      int size = is.available();

      for(int i=0; i< size; i++){
         System.out.print((char)is.read() + "  ");
      }
      is.close();
   }catch(IOException e){
      System.out.print("Exception");
   }
   }
}
```

上面的程序首先创建文件test.txt，并把给定的数字以二进制形式写进该文件，同时输出到控制台上。

以上代码由于是二进制写入，可能存在乱码，你可以使用以下代码实例来解决乱码问题：

```
//文件名 :fileStreamTest2.java
import java.io.*;

public class fileStreamTest2{
    public static void main(String[] args) throws IOException {

        File f = new File("a.txt");
        FileOutputStream fop = new FileOutputStream(f);
        // 构建FileOutputStream对象,文件不存在会自动新建

        OutputStreamWriter writer = new OutputStreamWriter(fop, "U
        // 构建OutputStreamWriter对象,参数可以指定编码,默认为操作系统默认

        writer.append("中文输入");
        // 写入到缓冲区

        writer.append("\r\n");
        //换行

        writer.append("English");
        // 刷新缓存冲,写入到文件,如果下面已经没有写入的内容了,直接close也会

        writer.close();
        //关闭写入流,同时会把缓冲区内容写入文件,所以上面的注释掉

        fop.close();
        // 关闭输出流,释放系统资源

        FileInputStream fip = new FileInputStream(f);
        // 构建FileInputStream对象

        InputStreamReader reader = new InputStreamReader(fip, "UTF
        // 构建InputStreamReader对象,编码与写入相同

        StringBuffer sb = new StringBuffer();
        while (reader.ready()) {
            sb.append((char) reader.read());
            // 转成char加到StringBuffer对象中
        }
        System.out.println(sb.toString());
        reader.close();
        // 关闭读取流

        fip.close();
        // 关闭输入流,释放系统资源

    }
}
```

# 文件和I/O

还有一些关于文件和I/O的类，我们也需要知道：

- File Class(类)
- FileReader Class(类)
- FileWriter Class(类)

# Java中的目录

## 创建目录：

File类中有两个方法可以用来创建文件夹：

- **mkdir( )**方法创建一个文件夹，成功则返回true，失败则返回false。失败表明File对象指定的路径已经存在，或者由于整个路径还不存在，该文件夹不能被创建。
- **mkdirs()**方法创建一个文件夹和它的所有父文件夹。

下面的例子创建 "/tmp/user/java/bin"文件夹：

```
import java.io.File;

public class CreateDir {
   public static void main(String args[]) {
      String dirname = "/tmp/user/java/bin";
      File d = new File(dirname);
      // 现在创建目录
      d.mkdirs();
  }
}
```

编译并执行上面代码来创建目录"/tmp/user/java/bin"。

注意：Java在UNIX和Windows自动按约定分辨文件路径分隔符。如果你在Windows版本的Java中使用分隔符(/)，路径依然能够被正确解析。

## 读取目录

一个目录其实就是一个File对象，它包含其他文件和文件夹。

如果创建一个File对象并且它是一个目录，那么调用isDirectory( )方法会返回true。

可以通过调用该对象上的list()方法，来提取它包含的文件和文件夹的列表。

下面展示的例子说明如何使用list()方法来检查一个文件夹中包含的内容：

```
import java.io.File;

public class DirList {
   public static void main(String args[]) {
      String dirname = "/tmp";
      File f1 = new File(dirname);
      if (f1.isDirectory()) {
         System.out.println( "Directory of " + dirname);
         String s[] = f1.list();
         for (int i=0; i < s.length; i++) {
            File f = new File(dirname + "/" + s[i]);
            if (f.isDirectory()) {
               System.out.println(s[i] + " is a directory");
            } else {
               System.out.println(s[i] + " is a file");
            }
         }
      } else {
         System.out.println(dirname + " is not a directory");
    }
  }
}
```

以上实例编译运行结果如下：

```
Directory of /tmp
bin is a directory
lib is a directory
demo is a directory
test.txt is a file
README is a file
index.html is a file
include is a directory
```

# Java 异常处理

异常是程序中的一些错误，但并不是所有的错误都是异常，并且错误有时候是可以避免的。

比如说，你的代码少了一个分号，那么运行出来结果是提示是错误java.lang.Error；如果你用System.out.println(11/0)，那么你是因为你用0做了除数，会抛出java.lang.ArithmeticException的异常。

异常发生的原因有很多，通常包含以下几大类：

- 用户输入了非法数据。
- 要打开的文件不存在。
- 网络通信时连接中断，或者JVM内存溢出。

这些异常有的是因为用户错误引起，有的是程序错误引起的，还有其它一些是因为物理错误引起的。-

要理解Java异常处理是如何工作的，你需要掌握以下三种类型的异常：

- 检查性异常：最具代表的检查性异常是用户错误或问题引起的异常，这是程序员无法预见的。例如要打开一个不存在文件时，一个异常就发生了，这些异常在编译时不能被简单地忽略。
- 运行时异常：运行时异常是可能被程序员避免的异常。与检查性异常相反，运行时异常可以在编译时被忽略。
- 错误：错误不是异常，而是脱离程序员控制的问题。错误在代码中通常被忽略。例如，当栈溢出时，一个错误就发生了，它们在编译也检查不到的。

# Exception类的层次

所有的异常类是从java.lang.Exception类继承的子类。

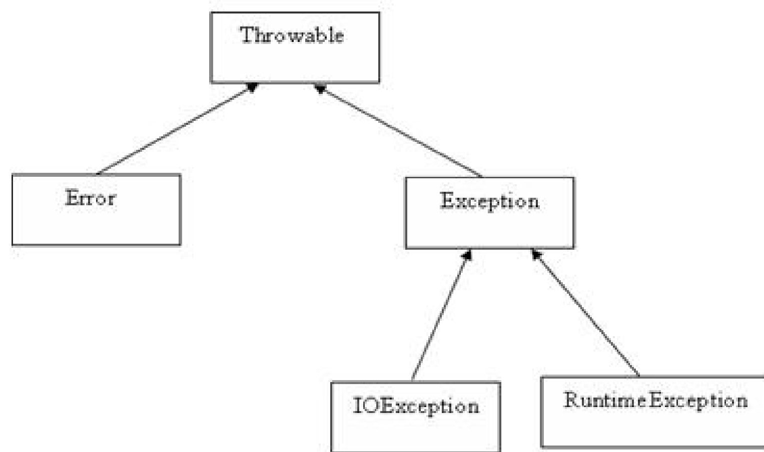
Exception类是Throwable类的子类。除了Exception类外，Throwable还有一个子类Error 。

Java程序通常不捕获错误。错误一般发生在严重故障时，它们在Java程序处理的范畴之外。

Error用来指示运行时环境发生的错误。

例如，JVM内存溢出。一般地，程序不会从错误中恢复。

异常类有两个主要的子类：IOException类和RuntimeException类。




在Java 内置类中(接下来会说明)，有大部分常用检查性和非检查性异常。

## Java 内置异常类

Java 语言定义了一些异常类在java.lang标准包中。

标准运行时异常类的子类是最常见的异常类。由于java.lang包是默认加载到所有的Java程序的，所以大部分从运行时异常类继承而来的异常都可以直接使用。

Java根据各个类库也定义了一些其他的异常，下面的表中列出了Java的非检查性异常。

| 异常 | 描述 |
|---|---|
| ArithmeticException | 当出现异常的运算条件时，抛出此异常。例如，一个整数"除以零"时，抛出此类的一个实例。 |
| ArrayIndexOutOfBoundsException | 用非法索引访问数组时抛出的异常。如果索引为负或大于等于数组大小，则该索引为非法索引。 |
| ArrayStoreException | 试图将错误类型的对象存储到一个对象数组时抛出的异常。 |
| ClassCastException | 当试图将对象强制转换为不是实例的子类时，抛出该异常。 |
| IllegalArgumentException | 抛出的异常表明向方法传递了一个不合法或不正确的参数。 |
| IllegalMonitorStateException | 抛出的异常表明某一线程已经试图等待对象的监视器，或者试图通知其他正在等待对象的监视器而本身没有指定监视器的线程。 |
| IllegalStateException | 在非法或不适当的时间调用方法时产生的信号。换句话说，即 Java 环境或 Java 应用程序没有处于请求操作所要求的适当状态下。 |
| IllegalThreadStateException | 线程没有处于请求操作所要求的适当状态时抛出的异常。 |
| IndexOutOfBoundsException | 指示某排序索引（例如对数组、字符串或向量的排序）超出范围时抛出。 |
| NegativeArraySizeException | 如果应用程序试图创建大小为负的数组，则抛出该异常。 |
| NullPointerException | 当应用程序试图在需要对象的地方使用 `null` 时，抛出该异常 |
| NumberFormatException | 当应用程序试图将字符串转换成一种数值类型，但该字符串不能转换为适当格式时，抛出该异常。 |
| SecurityException | 由安全管理器抛出的异常，指示存在安全侵犯。 |
| StringIndexOutOfBoundsException | 此异常由 `String` 方法抛出，指示索引或者为负，或者超出字符串的大小。 |
| UnsupportedOperationException | 当不支持请求的操作时，抛出该异常。 |

下面的表中列出了Java定义在java.lang包中的检查性异常类。

| 异常 | 描述 |
| --- | --- |
| ClassNotFoundException | 应用程序试图加载类时，找不到相应的类，抛出该异常。 |
| CloneNotSupportedException | 当调用 `Object` 类中的 `clone` 方法克隆对象，但该对象的类无法实现 `Cloneable` 接口时，抛出该异常。 |
| IllegalAccessException | 拒绝访问一个类的时候，抛出该异常。 |
| InstantiationException | 当试图使用 `Class` 类中的 `newInstance` 方法创建一个类的实例，而指定的类对象因为是一个接口或是一个抽象类而无法实例化时，抛出该异常。 |
| InterruptedException | 一个线程被另一个线程中断，抛出该异常。 |
| NoSuchFieldException | 请求的变量不存在 |
| NoSuchMethodException | 请求的方法不存在 |

# 异常方法

下面的列表是Throwable 类的主要方法:

| 方法 | 说明 |
| --- | --- |
| **public String getMessage()** | 返回关于发生的异常的详细信息。这个消息在Throwable 类的构造函数中初始化了。 |
| **public Throwable getCause()** | 返回一个Throwable 对象代表异常原因。 |
| **public String toString()** | 使用getMessage()的结果返回类的串级名字。 |
| **public void printStackTrace()** | 打印toString()结果和栈层次到System.err，即错误输出流。 |
| **public StackTraceElement [] getStackTrace()** | 返回一个包含堆栈层次的数组。下标为0的元素代表栈顶，最后一个元素代表方法调用堆栈的栈底。 |
| **public Throwable fillInStackTrace()** | 用当前的调用栈层次填充Throwable 对象栈层次，添加到栈层次任何先前信息中。 |

# 捕获异常

使用try和catch关键字可以捕获异常。try/catch代码块放在异常可能发生的地方。

try/catch代码块中的代码称为保护代码，使用 try/catch的语法如下：

```
try
{
   // 程序代码
}catch(ExceptionName e1)
{
   //Catch 块
}
```

Catch语句包含要捕获异常类型的声明。当保护代码块中发生一个异常时，try后面的catch块就会被检查。

如果发生的异常包含在catch块中，异常会被传递到该catch块，这和传递一个参数到方法是一样。

## 实例

下面的例子中声明有两个元素的一个数组，当代码试图访问数组的第三个元素的时候就会抛出一个异常。

```
// 文件名 : ExcepTest.java
import java.io.*;
public class ExcepTest{

   public static void main(String args[]){
      try{
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

以上代码编译运行输出结果如下：

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

# 多重捕获块

一个try代码块后面跟随多个catch代码块的情况就叫多重捕获。

多重捕获块的语法如下所示：

```
try{
   // 程序代码
}catch(异常类型1 异常的变量名1){
   // 程序代码
}catch(异常类型2 异常的变量名2){
   // 程序代码
}catch(异常类型2 异常的变量名2){
   // 程序代码
}
```

上面的代码段包含了3个catch块。

可以在ry语句后面添加任意数量的catch块。

如果保护代码中发生异常，异常被抛给第一个catch块。

如果抛出异常的数据类型与ExceptionType1匹配，它在这里就会被捕获。

如果不匹配，它会被传递给第二个catch块。

如此，直到异常被捕获或者通过所有的catch块。

### 实例

该实例展示了怎么使用多重try/catch。

```
try
{
   file = new FileInputStream(fileName);
   x = (byte) file.read();
}catch(IOException i)
{
   i.printStackTrace();
   return -1;
}catch(FileNotFoundException f) //Not valid!
{
   f.printStackTrace();
   return -1;
}
```

## throws/throw关键字：

如果一个方法没有捕获一个检查性异常，那么该方法必须使用throws 关键字来声明。throws关键字放在方法签名的尾部。

也可以使用throw关键字抛出一个异常，无论它是新实例化的还是刚捕获到的。

下面方法的声明抛出一个RemoteException异常：

```
import java.io.*;
public class className
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```

一个方法可以声明抛出多个异常，多个异常之间用逗号隔开。

例如，下面的方法声明抛出RemoteException和InsufficientFundsException：

```
import java.io.*;
public class className
{
   public void withdraw(double amount) throws RemoteException,
                              InsufficientFundsException
   {
       // Method implementation
   }
   //Remainder of class definition
}
```

# finally关键字

finally关键字用来创建在try代码块后面执行的代码块。

无论是否发生异常，finally代码块中的代码总会被执行。

在finally代码块中，可以运行清理类型等收尾善后性质的语句。

finally代码块出现在catch代码块最后，语法如下：

```
try{
   // 程序代码
}catch(异常类型1 异常的变量名1){
   // 程序代码
}catch(异常类型2 异常的变量名2){
   // 程序代码
}finally{
   // 程序代码
}
```

## 实例

```
 public class ExcepTest{

   public static void main(String args[]){
      int a[] = new int[2];
      try{
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      finally{
         a[0] = 6;
         System.out.println("First element value: " +a[0]);
         System.out.println("The finally statement is executed");
      }
   }
}
```

以上实例编译运行结果如下：

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

注意下面事项：

- catch不能独立于try存在。
- 在try/catch后面添加finally块并非强制性要求的。
- try代码后不能既没catch块也没finally块。
- try, catch, finally块之间不能添加任何代码。

# 声明自定义异常

在Java中你可以自定义异常。编写自己的异常类时需要记住下面的几点。

- 所有异常都必须是Throwable的子类。
- 如果希望写一个检查性异常类，则需要继承Exception类。
- 如果你想写一个运行时异常类，那么需要继承RuntimeException 类。

可以像下面这样定义自己的异常类：

```
class MyException extends Exception{
}
```

只继承Exception 类来创建的异常类是检查性异常类。

下面的InsufficientFundsException类是用户定义的异常类，它继承自Exception。

一个异常类和其它任何类一样，包含有变量和方法。

## 实例

```
// 文件名InsufficientFundsException.java
import java.io.*;

public class InsufficientFundsException extends Exception
{
   private double amount;
   public InsufficientFundsException(double amount)
   {
      this.amount = amount;
   }
   public double getAmount()
   {
      return amount;
   }
}
```

为了展示如何使用我们自定义的异常类，

在下面的CheckingAccount 类中包含一个withdraw()方法抛出一个InsufficientFundsException异常。

```
// 文件名称 CheckingAccount.java
import java.io.*;

public class CheckingAccount
{
   private double balance;
   private int number;
   public CheckingAccount(int number)
   {
      this.number = number;
   }
   public void deposit(double amount)
   {
      balance += amount;
   }
   public void withdraw(double amount) throws
                              InsufficientFundsException
   {
      if(amount <= balance)
      {
         balance -= amount;
      }
      else
      {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }
   public double getBalance()
   {
      return balance;
   }
   public int getNumber()
   {
      return number;
   }
}
```

下面的BankDemo程序示范了如何调用CheckingAccount类的deposit() 和withdraw()方法。

```
//文件名称 BankDemo.java
public class BankDemo
{
   public static void main(String [] args)
   {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);
      try
      {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      }catch(InsufficientFundsException e)
      {
         System.out.println("Sorry, but you are short $"
                                  + e.getAmount());
         e.printStackTrace();
      }
    }
}
```

编译上面三个文件，并运行程序BankDemo，得到结果如下所示：

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
        at CheckingAccount.withdraw(CheckingAccount.java:25)
        at BankDemo.main(BankDemo.java:13)
```

# 通用异常

在Java中定义了两种类型的异常和错误。

- **JVM(Java**虚拟机)**异常：**由JVM抛出的异常或错误。例如：NullPointerException类，ArrayIndexOutOfBoundsException类，ClassCastException类。
- 程序级异常：由程序或者API程序抛出的异常。例如IllegalArgumentException类，IllegalStateException类。

# Java 面向对象

# Java 继承

继承是java面向对象编程技术的一块基石，因为它允许创建分等级层次的类。继承可以理解为一个对象从另一个对象获取属性的过程。

如果类A是类B的父类，而类B是类C的父类，我们也称C是A的子类，类C是从类A继承而来的。在Java中，类的继承是单一继承，也就是说，一个子类只能拥有一个父类

继承中最常使用的两个关键字是extends和implements。

这两个关键字的使用决定了一个对象和另一个对象是否是IS-A(是一个)关系。

通过使用这两个关键字，我们能实现一个对象获取另一个对象的属性。

所有Java的类均是由java.lang.Object类继承而来的，所以Object是所有类的祖先类，而除了Object外，所有类必须有一个父类。

通过过extends关键字可以申明一个类是继承另外一个类而来的，一般形式如下：

```
// A.java
public class A {
    private int i;
    protected int j;

    public void func() {

    }
}

// B.java
public class B extends A {
}
```

以上的代码片段说明，B由A继承而来的，B是A的子类。而A是Object的子类，这里可以不显示地声明。

作为子类，B的实例拥有A所有的成员变量，但对于private的成员变量B却没有访问权限，这保障了A的封装性。

# IS-A关系

IS-A就是说:一个对象是另一个对象的一个分类。

下面是使用关键字extends实现继承。

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

基于上面的例子，以下说法是正确的：

- Animal类是Mammal类的父类。
- Animal类是Reptile类的父类。
- Mammal类和Reptile类是Animal类的子类。
- Dog类既是Mammal类的子类又是Animal类的子类。

分析以上示例中的IS-A关系，如下：

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal

因此 : Dog IS-A Animal

通过使用关键字**extends**，子类可以继承父类的除private属性外所有的属性。

我们通过使用instanceof 操作符，能够确定Mammal IS-A Animal

## 实例

```
public class Dog extends Mammal{

   public static void main(String args[]){

      Animal a = new Animal();
      Mammal m = new Mammal();
      Dog d = new Dog();

      System.out.println(m instanceof Animal);
      System.out.println(d instanceof Mammal);
      System.out.println(d instanceof Animal);
   }
}
```

以上实例编译运行结果如下：

```
true
true
true
```

介绍完**extends**关键字之后，我们再来看下**implements**关键字是怎样使用来表示IS-A关系。

**Implements**关键字使用在类继承接口的情况下， 这种情况不能使用关键字**extends**。

## 实例

```
public interface Animal {}

public class Mammal implements Animal{
}

public class Dog extends Mammal{
}
```

# instanceof 关键字

可以使用 **instanceof** 运算符来检验Mammal和dog对象是否是Animal类的一个实例。

```
interface Animal{}

class Mammal implements Animal{}

public class Dog extends Mammal{
   public static void main(String args[]){

      Mammal m = new Mammal();
      Dog d = new Dog();

      System.out.println(m instanceof Animal);
      System.out.println(d instanceof Mammal);
      System.out.println(d instanceof Animal);
   }
}
```

以上实例编译运行结果如下：

```
true
true
true
```

# HAS-A 关系

HAS-A代表类和它的成员之间的从属关系。这有助于代码的重用和减少代码的错误。

## 例子

```
public class Vehicle{}
public class Speed{}
public class Van extends Vehicle{
    private Speed sp;
}
```

Van类和Speed类是HAS-A关系(Van有一个Speed)，这样就不用将Speed类的全部代码粘贴到Van类中了，并且Speed类也可以重复利用于多个应用程序。

在面向对象特性中，用户不必担心类的内部怎样实现。

Van类将实现的细节对用户隐藏起来，因此，用户只需要知道怎样调用Van类来完成某一功能，而不必知道Van类是自己来做还是调用其他类来做这些工作。

Java只支持单继承，也就是说，一个类不能继承多个类。

下面的做法是不合法的：

```
public class extends Animal, Mammal{}
```

Java只支持单继承（继承基本类和抽象类），但是我们可以用接口来实现（多继承接口来实现）,脚本结构如：

```
public class Apple extends Fruit implements Fruit1, Fruit2{}
```

一般我们继承基本类和抽象类用extends关键字，实现接口类的继承用implements关键字。

# Java 重写(Override)与重载(Overload)

## 重写(Override)

重写是子类对父类的允许访问的方法的实现过程进行重新编写！返回值和形参都不能改变。即外壳不变，核心重写！

重写的好处在于子类可以根据需要，定义特定于自己的行为。

也就是说子类能够根据需要实现父类的方法。

在面向对象原则里，重写意味着可以重写任何现有方法。实例如下：

```
class Animal{

   public void move(){
      System.out.println("动物可以移动");
   }
}

class Dog extends Animal{

   public void move(){
      System.out.println("狗可以跑和走");
   }
}

public class TestDog{

   public static void main(String args[]){
      Animal a = new Animal(); // Animal 对象
      Animal b = new Dog(); // Dog 对象

      a.move();// 执行 Animal 类的方法

      b.move();//执行 Dog 类的方法
   }
}
```

以上实例编译运行结果如下：

```
动物可以移动
狗可以跑和走
```

在上面的例子中可以看到，尽管b属于Animal类型，但是它运行的是Dog类的move方法。

这是由于在编译阶段，只是检查参数的引用类型。

然而在运行时，Java虚拟机(JVM)指定对象的类型并且运行该对象的方法。

因此在上面的例子中，之所以能编译成功，是因为Animal类中存在move方法，然而运行时，运行的是特定对象的方法。

思考以下例子：

```
class Animal{

   public void move(){
      System.out.println("动物可以移动");
   }
}

class Dog extends Animal{

   public void move(){
      System.out.println("狗可以跑和走");
   }
   public void bark(){
      System.out.println("狗可以吠叫");
   }
}

public class TestDog{

   public static void main(String args[]){
      Animal a = new Animal(); // Animal 对象
      Animal b = new Dog(); // Dog 对象

      a.move();// 执行 Animal 类的方法
      b.move();//执行 Dog 类的方法
      b.bark();
   }
}
```

以上实例编译运行结果如下：

```
TestDog.java:30: cannot find symbol
symbol  : method bark()
location: class Animal
                b.bark();
                 ^
```

该程序将抛出一个编译错误，因为b的引用类型Animal没有bark方法。

## 方写重写的规则

- 参数列表必须完全与被重写方法的相同；
- 返回类型必须完全与被重写方法的返回类型相同；
- 访问权限不能比父类中被重写的方法的访问权限更高。例如：如果父类的一个方法被声明为public，那么在子类中重写该方法就不能声明为protected。
- 父类的成员方法只能被它的子类重写。
- 声明为final的方法不能被重写。
- 声明为static的方法不能被重写，但是能够被再次声明。
- 如果一个方法不能被继承，那么该方法不能被重写。
- 子类和父类在同一个包中，那么子类可以重写父类所有方法，除了声明为private和final的方法。
- 子类和父类不在同一个包中，那么子类只能够重写父类的声明为public和protected的非final方法。
- 重写的方法能够抛出任何非强制异常，无论被重写的方法是否抛出异常。但是，重写的方法不能抛出新的强制性异常，或者比被重写方法声明的更广泛的强制性异常，反之则可以。
- 构造方法不能被重写。
- 如果不能继承一个方法，则不能重写这个方法。

## Super关键字的使用

当需要在子类中调用父类的被重写方法时，要使用super关键字。

```
class Animal{

   public void move(){
      System.out.println("动物可以移动);
   }
}

class Dog extends Animal{

   public void move(){
      super.move(); // 应用super类的方法
      System.out.println("狗可以跑和走");
   }
}

public class TestDog{

   public static void main(String args[]){

      Animal b = new Dog(); /
      b.move(); //执行 Dog类的方法

   }
}
```

以上实例编译运行结果如下：

```
动物可以移动
狗可以跑和走
```

# 重载(Overload)

重载(overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型呢？可以相同也可以不同。

每个重载的方法（或者构造函数）都必须有一个独一无二的参数类型列表。

只能重载构造函数

重载规则

- 被重载的方法必须改变参数列表；
- 被重载的方法可以改变返回类型；
- 被重载的方法可以改变访问修饰符；
- 被重载的方法可以声明新的或更广的检查异常；
- 方法能够在同一个类中或者在一个子类中被重载。

## 实例

```
public class Overloading {

    public int test(){
        System.out.println("test1");
        return 1;
    }

    public void test(int a){
        System.out.println("test2");
    }

    //以下两个参数类型顺序不同
    public String test(int a,String s){
        System.out.println("test3");
        return "returntest3";
    }

    public String test(String s,int a){
        System.out.println("test4");
        return "returntest4";
    }

    public static void main(String[] args){
        Overloading o = new Overloading();
        System.out.println(o.test());
        o.test(1);
        System.out.println(o.test(1,"test3"));
        System.out.println(o.test("test4",1));
    }
```

## 重写与重载之间的区别

| 区别点 | 重载方法 | 重写方法 |
| --- | --- | --- |
| 参数列表 | 必须修改 | 一定不能修改 |
| 返回类型 | 可以修改 | 一定不能修改 |
| 异常 | 可以修改 | 可以减少或删除，一定不能抛出新的或者更广的异常 |
| 访问 | 可以修改 | 一定不能做更严格的限制（可以降低限制） |

# Java 多态

多态是同一个行为具有多个不同表现形式或形态的能力。

多态性是对象多种表现形式的体现。

比如我们说"宠物"这个对象，它就有很多不同的表达或实现，比如有小猫、小狗、蜥蜴等等。那么我到宠物店说"请给我一只宠物"，服务员给我小猫、小狗或者蜥蜴都可以，我们就说"宠物"这个对象就具备多态性。

接下来让我们通过实例来了解Java的多态。

## 例子

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

因为Deer类具有多重继承，所以它具有多态性。以上实例解析如下：

- 一个 Deer IS-A（是一个） Animal
- 一个 Deer IS-A（是一个） Vegetarian
- 一个 Deer IS-A（是一个） Deer
- 一个 Deer IS-A（是一个）Object

在Java中，所有的对象都具有多态性，因为任何对象都能通过IS-A测试的类型和Object类。

访问一个对象的唯一方法就是通过引用型变量。

引用型变量只能有一种类型，一旦被声明，引用型变量的类型就不能被改变了。

引用型变量不仅能够被重置为其他对象，前提是这些对象没有被声明为final。还可以引用和它类型相同的或者相兼容的对象。它可以声明为类类型或者接口类型。

当我们将引用型变量应用于Deer对象的引用时，下面的声明是合法的：

```
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

所有的引用型变量d,a,v,o都指向堆中相同的Deer对象。

# 虚方法

我们将介绍在Java中，当设计类时，被重载的方法的行为怎样影响多态性。

我们已经讨论了方法的重载，也就是子类能够重载父类的方法。

当子类对象调用重载的方法时，调用的是子类的方法，而不是父类中被重载的方法。

要想调用父类中被重载的方法，则必须使用关键字super。

```
/* 文件名 : Employee.java */
public class Employee
{
   private String name;
   private String address;
   private int number;
   public Employee(String name, String address, int number)
   {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }
   public void mailCheck()
   {
      System.out.println("Mailing a check to " + this.name
       + " " + this.address);
   }
   public String toString()
   {
      return name + " " + address + " " + number;
   }
   public String getName()
   {
      return name;
   }
   public String getAddress()
   {
      return address;
   }
   public void setAddress(String newAddress)
   {
      address = newAddress;
   }
   public int getNumber()
   {
     return number;
   }
}
```

假设下面的类继承Employee类：

```
/* 文件名 : Salary.java */
public class Salary extends Employee
{
   private double salary; //Annual salary
   public Salary(String name, String address, int number, double
      salary)
   {
       super(name, address, number);
       setSalary(salary);
   }
   public void mailCheck()
   {
       System.out.println("Within mailCheck of Salary class ");
       System.out.println("Mailing check to " + getName()
       + " with salary " + salary);
   }
   public double getSalary()
   {
       return salary;
   }
   public void setSalary(double newSalary)
   {
       if(newSalary >= 0.0)
       {
          salary = newSalary;
       }
   }
   public double computePay()
   {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
}
```

现在我们仔细阅读下面的代码，尝试给出它的输出结果：

```
/* 文件名 : VirtualDemo.java */
public class VirtualDemo
{
   public static void main(String [] args)
   {
      Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 360
      Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.0
      System.out.println("Call mailCheck using Salary reference --"
      s.mailCheck();
      System.out.println("\n Call mailCheck using Employee referenc
      e.mailCheck();
    }
}
```

以上实例编译运行结果如下：

```
Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0
```

例子中，我们实例化了两个Salary对象。一个使用Salary引用s，另一个使用Employee引用。

编译时，编译器检查到mailCheck()方法在Salary类中的声明。

在调用s.mailCheck()时，Java虚拟机(JVM)调用Salary类的mailCheck()方法。

因为e是Employee的引用，所以调用e的mailCheck()方法则有完全不同的结果。

当编译器检查e.mailCheck()方法时，编译器检查到Employee类中的mailCheck()方法。

在编译的时候，编译器使用Employee类中的mailCheck()方法验证该语句， 但是在运行的时候，Java虚拟机(JVM)调用的是Salary类中的mailCheck()方法。

该行为被称为虚拟方法调用，该方法被称为虚拟方法。

Java中所有的方法都能以这种方式表现，借此，重写的方法能在运行时调用，不管编译的时候源代码中引用变量是什么数据类型。

# Java 抽象类

在面向对象的概念中，所有的对象都是通过类来描绘的，但是反过来，并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。

抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。

由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。也是因为这个原因，通常在设计阶段决定要不要设计抽象类。

父类包含了子类集合的常见的方法，但是由于父类本身是抽象的，所以不能使用这些方法。

## 抽象类

在Java语言中使用abstract class来定义抽象类。如下实例：

```
/* 文件名 : Employee.java */
public abstract class Employee
{
   private String name;
   private String address;
   private int number;
   public Employee(String name, String address, int number)
   {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }
   public double computePay()
   {
     System.out.println("Inside Employee computePay");
     return 0.0;
   }
   public void mailCheck()
   {
      System.out.println("Mailing a check to " + this.name
       + " " + this.address);
   }
   public String toString()
   {
      return name + " " + address + " " + number;
   }
   public String getName()
   {
      return name;
   }
   public String getAddress()
   {
      return address;
   }
   public void setAddress(String newAddress)
   {
      address = newAddress;
   }
   public int getNumber()
   {
     return number;
   }
}
```

注意到该Employee类没有什么不同，尽管该类是抽象类，但是它仍然有3个成员变量，7个成员方法和1个构造方法。 现在如果你尝试如下的例子：

```
/* 文件名 : AbstractDemo.java */
public class AbstractDemo
{
   public static void main(String [] args)
   {
      /* 以下是不允许的，会引发错误 */
      Employee e = new Employee("George W.", "Houston, TX", 43);

      System.out.println("\n Call mailCheck using Employee referenc
      e.mailCheck();
    }
}
```

当你尝试编译AbstractDemo类时，会产生如下错误：

```
Employee.java:46: Employee is abstract; cannot be instantiated
      Employee e = new Employee("George W.", "Houston, TX", 43);
                   ^
1 error
```

## 继承抽象类

我们能通过一般的方法继承Employee类：

```
/* 文件名 : Salary.java */
public class Salary extends Employee
{
   private double salary; //Annual salary
   public Salary(String name, String address, int number, double
      salary)
   {
       super(name, address, number);
       setSalary(salary);
   }
   public void mailCheck()
   {
       System.out.println("Within mailCheck of Salary class ");
       System.out.println("Mailing check to " + getName()
       + " with salary " + salary);
   }
   public double getSalary()
   {
       return salary;
   }
   public void setSalary(double newSalary)
   {
       if(newSalary >= 0.0)
       {
          salary = newSalary;
       }
   }
   public double computePay()
   {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
}
```

尽管我们不能实例化一个Employee类的对象，但是如果我们实例化一个Salary类对象，该对象将从Employee类继承3个成员变量和7个成员方法。

```
/* 文件名 : AbstractDemo.java */
public class AbstractDemo
{
   public static void main(String [] args)
   {
      Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 360
      Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.0

      System.out.println("Call mailCheck using Salary reference --"
      s.mailCheck();

      System.out.println("\n Call mailCheck using Employee referenc
      e.mailCheck();
    }
}
```

以上程序编译运行结果如下：

```
Constructing an Employee
Constructing an Employee
Call mailCheck using  Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.
```

# 抽象方法

如果你想设计这样一个类，该类包含一个特别的成员方法，该方法的具体实现由它的子类确定，那么你可以在父类中声明该方法为抽象方法。

Abstract关键字同样可以用来声明抽象方法，抽象方法只包含一个方法名，而没有方法体。

抽象方法没有定义，方法名后面直接跟一个分号，而不是花括号。

```
public abstract class Employee
{
   private String name;
   private String address;
   private int number;

   public abstract double computePay();

   //其余代码
}
```

声明抽象方法会造成以下两个结果：

- 如果一个类包含抽象方法，那么该类必须是抽象类。
- 任何子类必须重写父类的抽象方法，或者声明自身为抽象类。

继承抽象方法的子类必须重载该方法。否则，该子类也必须声明为抽象类。最终，必须有子类实现该抽象方法，否则，从最初的父类到最终的子类都不能用来实例化对象。

如果Salary类继承了Employee类，那么它必须实现computePay()方法：

```
/* 文件名 : Salary.java */
public class Salary extends Employee
{
   private double salary; // Annual salary

   public double computePay()
   {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }

   //其余代码
}
```

# java 封装

在面向对象程式设计方法中，封装（英语：Encapsulation）是指，一种将抽象性函式接口的实作细节部份包装、隐藏起来的方法。

封装可以被认为是一个保护屏障，防止该类的代码和数据被外部类定义的代码随机访问。

要访问该类的代码和数据，必须通过严格的接口控制。

封装最主要的功能在于我们能修改自己的实现代码，而不用修改那些调用我们代码的程序片段。

适当的封装可以让程式码更容易理解与维护，也加强了程式码的安全性。

## 实例

让我们来看一个java封装类的例子：

```
/* 文件名: EncapTest.java */
public class EncapTest{

   private String name;
   private String idNum;
   private int age;

   public int getAge(){
      return age;
   }

   public String getName(){
      return name;
   }

   public String getIdNum(){
      return idNum;
   }

   public void setAge( int newAge){
      age = newAge;
   }

   public void setName(String newName){
      name = newName;
   }

   public void setIdNum( String newId){
      idNum = newId;
   }
}
```

以上实例中public方法是外部类访问该类成员变量的入口。

通常情况下，这些方法被称为getter和setter方法。

因此，任何要访问类中私有成员变量的类都要通过这些getter和setter方法。

通过如下的例子说明EncapTest类的变量怎样被访问：

```
/* F文件名 : RunEncap.java */
public class RunEncap{

   public static void main(String args[]){
      EncapTest encap = new EncapTest();
      encap.setName("James");
      encap.setAge(20);
      encap.setIdNum("12343ms");

      System.out.print("Name : " + encap.getName()+
                             " Age : "+ encap.getAge());
    }
}
```

以上代码编译运行结果如下:

```
Name : James Age : 20
```

# Java 接口

接口（英文：Interface），在JAVA编程语言中是一个抽象类型，是抽象方法的集合，接口通常以interface来声明。一个类通过继承接口的方式，从而来继承接口的抽象方法。

接口并不是类，编写接口的方式和类很相似，但是它们属于不同的概念。类描述对象的属性和方法。接口则包含类要实现的方法。

除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。

接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在Java中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。

接口与类相似点：

- 一个接口可以有多个方法。
- 接口文件保存在.java结尾的文件中，文件名使用接口名。
- 接口的字节码文件保存在.class结尾的文件中。
- 接口相应的字节码文件必须在与包名称相匹配的目录结构中。

接口与类的区别：

- 接口不能用于实例化对象。
- 接口没有构造方法。
- 接口中所有的方法必须是抽象方法。
- 接口不能包含成员变量，除了static和final变量。
- 接口不是被类继承了，而是要被类实现。
- 接口支持多重继承。

## 接口的声明

接口的声明语法格式如下：

```
[可见度] interface 接口名称 [extends 其他的类名] {
        // 声明变量
        // 抽象方法
}
```

Interface关键字用来声明一个接口。下面是接口声明的一个简单例子。

```
/* 文件名 : NameOfInterface.java */
import java.lang.*;
//引入包

public interface NameOfInterface
{
   //任何类型 final, static 字段
   //抽象方法
}
```

接口有以下特性：

- 接口是隐式抽象的，当声明一个接口的时候，不必使用**abstract**关键字。
- 接口中每一个方法也是隐式抽象的，声明时同样不需要**abstract**关键子。
- 接口中的方法都是公有的。

### 实例

```
/* 文件名 : Animal.java */
interface Animal {

   public void eat();
   public void travel();
}
```

## 接口的实现

当类实现接口的时候，类要实现接口中所有的方法。否则，类必须声明为抽象的类。

类使用implements关键字实现接口。在类声明中，Implements关键字放在class声明后面。

实现一个接口的语法，可以使用这个公式：

```
... implements 接口名称[, 其他接口, 其他接口..., ...] ...
```

### 实例

```
/* 文件名 : MammalInt.java */
public class MammalInt implements Animal{

   public void eat(){
      System.out.println("Mammal eats");
   }

   public void travel(){
      System.out.println("Mammal travels");
   }

   public int noOfLegs(){
      return 0;
   }

   public static void main(String args[]){
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

以上实例编译运行结果如下:

```
Mammal eats
Mammal travels
```

重写接口中声明的方法时，需要注意以下规则：

- 类在实现接口的方法时，不能抛出强制性异常，只能在接口中，或者继承接口的抽象类中抛出该强制性异常。
- 类在重写方法时要保持一致的方法名，并且应该保持相同或者相兼容的返回值类型。
- 如果实现接口的类是抽象类，那么就没必要实现该接口的方法。

在实现接口的时候，也要注意一些规则：

- 一个类可以同时实现多个接口。
- 一个类只能继承一个类，但是能实现多个接口。
- 一个接口能继承另一个接口，这和类之间的继承比较相似。

# 接口的继承

一个接口能继承另一个接口，和类之间的继承方式比较相似。接口的继承使用extends关键字，子接口继承父接口的方法。

下面的Sports接口被Hockey和Football接口继承：

```
// 文件名: Sports.java
public interface Sports
{
   public void setHomeTeam(String name);
   public void setVisitingTeam(String name);
}

// 文件名: Football.java
public interface Football extends Sports
{
   public void homeTeamScored(int points);
   public void visitingTeamScored(int points);
   public void endOfQuarter(int quarter);
}

// 文件名: Hockey.java
public interface Hockey extends Sports
{
   public void homeGoalScored();
   public void visitingGoalScored();
   public void endOfPeriod(int period);
   public void overtimePeriod(int ot);
}
```

Hockey接口自己声明了四个方法，从Sports接口继承了两个方法，这样，实现Hockey接口的类需要实现六个方法。

相似的，实现Football接口的类需要实现五个方法，其中两个来自于Sports接口。

## 接口的多重继承

在Java中，类的多重继承是不合法，但接口允许多重继承，。

在接口的多重继承中extends关键字只需要使用一次，在其后跟着继承接口。 如下所示：

```
public interface Hockey extends Sports, Event
```

以上的程序片段是合法定义的子接口，与类不同的是，接口允许多重继承，而 Sports及 Event 可能定义或是继承相同的方法

## 标记接口

最常用的继承接口是没有包含任何方法的接口。

标识接口是没有任何方法和属性的接口.它仅仅表明它的类属于一个特定的类型,供其他代码来测试允许做一些事情。

标识接口作用：简单形象的说就是给某个对象打个标（盖个戳），使对象拥有某个或某些特权。

例如：java.awt.event包中的MouseListener接口继承的java.util.EventListener接口定义如下：

```
package java.util;
public interface EventListener
{}
```

没有任何方法的接口被称为标记接口。标记接口主要用于以下两种目的：

- 建立一个公共的父接口：

  正如EventListener接口，这是由几十个其他接口扩展的Java API，你可以使用一个标记接口来建立一组接口的父接口。例如：当一个接口继承了EventListener接口，Java虚拟机(JVM)就知道该接口将要被用于一个事件的代理方案。

- 向一个类添加数据类型：

  这种情况是标记接口最初的目的，实现标记接口的类不需要定义任何接口方法(因为标记接口根本就没有方法)，但是该类通过多态性变成一个接口类型。

# Java 包(package)

为了更好地组织类，Java提供了包机制，用于区别类名的命名空间。

包的作用

- 1 把功能相似或相关的类或接口组织在同一个包中，方便类的查找和使用。
- 2 如同文件夹一样，包也采用了树形目录的存储方式。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。因此，包可以避免名字冲突。
- 3 包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类。

Java使用包（package）这种机制是为了防止命名冲突，访问控制，提供搜索和定位类（class）、接口、枚举（enumerations）和注释（annotation）等。

包语句的语法格式为：

```
package pkg1[. pkg2[. pkg3…]];
```

例如,一个Something.java 文件它的内容

```
package net.java.util
public class Something{
   ...
}
```

那么它的路径应该是 net/java/Something.java 这样保存的。 package(包)的作用是把不同的java程序分类保存，更方便的被其他java程序调用。

一个包（package）可以定义为一组相互联系的类型（类、接口、枚举和注释），为这些类型提供访问保护和命名空间管理的功能。

以下是一些Java中的包：

- java.lang-打包基础的类
- java.io-包含输入输出功能的函数

开发者可以自己把一组类和接口等打包，并定义自己的package。而且在实际开发中这样做是值得提倡的，当你自己完成类的实现之后，将相关的类分组，可以让其他的编程者更容易地确定哪些类、接口、枚举和注释等是相关的。

由于package创建了新的命名空间（namespace），所以不会跟其他package中的任何名字产生命名冲突。使用包这种机制，更容易实现访问控制，并且让定位相关类更加简单。

# 创建包

创建package的时候，你需要为这个package取一个合适的名字。之后，如果其他的一个源文件包含了这个包提供的类、接口、枚举或者注释类型的时候，都必须将这个package的声明放在这个源文件的开头。

包声明应该在源文件的第一行，每个源文件只能有一个包声明，这个文件中的每个类型都应用于它。

如果一个源文件中没有使用包声明，那么其中的类，函数，枚举，注释等将被放在一个无名的包（unnamed package）中。

## 例子

让我们来看一个例子，这个例子创建了一个叫做animals的包。通常使用小写的字母来命名避免与类、接口名字的冲突。

在animals包中加入一个接口（interface）：

```
/* 文件名: Animal.java */
package animals;

interface Animal {
   public void eat();
   public void travel();
}
```

接下来，在同一个包中加入该接口的实现：

```
package animals;

/* 文件名 : MammalInt.java */
public class MammalInt implements Animal{

   public void eat(){
      System.out.println("Mammal eats");
   }

   public void travel(){
      System.out.println("Mammal travels");
   }

   public int noOfLegs(){
      return 0;
   }

   public static void main(String args[]){
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

然后，编译这两个文件，并把他们放在一个叫做animals的子目录中。 用下面的命令来运行：

```
$ mkdir animals
$ cp Animal.class  MammalInt.class animals
$ java animals/MammalInt
Mammal eats
Mammal travel
```

## import关键字

为了能够使用某一个包的成员，我们需要在 Java 程序中明确导入该包。使用"import"语句可完成此功能。

在 java 源文件中 import 语句应位于 package 语句之后，所有类的定义之前，可以没有，也可以有多条，其语法格式为：

```
import package1[.package2…].(classname|*);
```

如果在一个包中，一个类想要使用本包中的另一个类，那么该包名可以省略。

## 例子

下面的payroll包已经包含了Employee类，接下来向payroll包中添加一个Boss类。Boss类引用Employee类的时候可以不用使用payroll前缀，Boss类的实例如下。

```
package payroll;

public class Boss
{
   public void payEmployee(Employee e)
   {
      e.mailCheck();
   }
}
```

如果Boss类不在payroll包中又会怎样？Boss类必须使用下面几种方法之一来引用其他包中的类

使用类全名描述，例如：

```
payroll.Employee
```

用import关键字引入，使用通配符"*"

```
import payroll.*;
```

使用import关键字引入Employee类

```
import payroll.Employee;
```

注意：

类文件中可以包含任意数量的import声明。import声明必须在包声明之后，类声明之前。

# package的目录结构

类放在包中会有两种主要的结果：

- 包名成为类名的一部分，正如我们前面讨论的一样。
- 包名必须与相应的字节码所在的目录结构相吻合。

下面是管理你自己java中文件的一种简单方式：

将类、接口等类型的源码放在一个文本中，这个文件的名字就是这个类型的名字，并以.java作为扩展名。例如：

```
// 文件名 :  Car.java

package vehicle;

public class Car {
   // 类实现
}
```

接下来，把源文件放在一个目录中，这个目录要对应类所在包的名字。

```
....\vehicle\Car.java
```

现在，正确的类名和路径将会是如下样子：

- 类名 -> vehicle.Car
- 路径名 -> vehicle\Car.java (in windows)

通常，一个公司使用它互联网域名的颠倒形式来作为它的包名.例如：互联网域名是apple.com，所有的包名都以com.apple开头。包名中的每一个部分对应一个子目录。

例如：这个公司有一个com.apple.computers的包，这个包包含一个叫做Dell.java的源文件，那么相应的，应该有如下面的一连串子目录：

```
....\com\apple\computers\Dell.java
```

编译的时候，编译器为包中定义的每个类、接口等类型各创建一个不同的输出文件，输出文件的名字就是这个类型的名字，并加上.class作为扩展后缀。 例如：

```
// 文件名: Dell.java

package com.apple.computers;
public class Dell{

}
class Ups{

}
```

现在，我们用-d选项来编译这个文件，如下：

```
$javac -d . Dell.java
```

这样会像下面这样放置编译了的文件：

```
.\com\apple\computers\Dell.class.\com\apple\computers\Ups.class
```

你可以像下面这样来导入所有 \com\apple\computers\中定义的类、接口等：

```
import com.apple.computers.*;
```

编译之后的.class文件应该和.java源文件一样，它们放置的目录应该跟包的名字对应起来。但是，并不要求.class文件的路径跟相应的.java的路径一样。你可以分开来安排源码和类的目录。

```
<path-one>\sources\com\apple\computers\Dell.java
<path-two>\classes\com\apple\computers\Dell.class
```

这样，你可以将你的类目录分享给其他的编程人员，而不用透露自己的源码。用这种方法管理源码和类文件可以让编译器和java虚拟机（JVM）可以找到你程序中使用的所有类型。

类目录的绝对路径叫做class path。设置在系统变量CLASSPATH中。编译器和java虚拟机通过将package名字加到class path后来构造.class文件的路径。

<path- two>\classes是class path，package名字是com.apple.computers,而编译器和JVM会在 <path-two>\classes\com\apple\compters中找.class文件。

一个class path可能会包含好几个路径。多路径应该用分隔符分开。默认情况下，编译器和JVM查找当前目录。JAR文件按包含Java平台相关的类，所以他们的目录默认放在了class path中。

# 设置CLASSPATH系统变量

用下面的命令显示当前的CLASSPATH变量：

- Windows平台（DOS 命令行下）-> C:\> set CLASSPATH
- UNIX平台（Bourne shell下）-> % echo $CLASSPATH

删除当前CLASSPATH变量内容：

- Windows平台（DOS 命令行下）-> C:\> set CLASSPATH=
- UNIX平台（Bourne shell下）-> % unset CLASSPATH; export CLASSPATH

设置CLASSPATH变量:

- Windows平台（DOS 命令行下）-> set CLASSPATH=C:\users\jack\java\classes
- UNIX平台（Bourne shell下）-> % CLASSPATH=/home/jack/java/classes; export CLASSPATH

# Java 高级教程

# Java 数据结构

Java工具包提供了强大的数据结构。在Java中的数据结构主要包括以下几种接口和类：

- 枚举（Enumeration）
- 位集合（BitSet）
- 向量（Vector）
- 栈（Stack）
- 字典（Dictionary）
- 哈希表（Hashtable）
- 属性（Properties）

以上这些类是传统遗留的，在Java2中引入了一种新的框架-集合框架(Collection)，我们后面再讨论。

## 枚举（Enumeration）

枚举（Enumeration）接口虽然它本身不属于数据结构,但它在其他数据结构的范畴里应用很广。 枚举（The Enumeration）接口定义了一种从数据结构中取回连续元素的方式。

例如，枚举定义了一个叫nextElement 的方法，该方法用来得到一个包含多元素的数据结构的下一个元素。

关于枚举接口的更多信息，请参见枚举（Enumeration）。

## 位集合（BitSet）

位集合类实现了一组可以单独设置和清除的位或标志。

该类在处理一组布尔值的时候非常有用，你只需要给每个值赋值一"位"，然后对位进行适当的设置或清除，就可以对布尔值进行操作了。

关于该类的更多信息，请参见位集合（BitSet）。

## 向量（Vector）

向量（Vector）类和传统数组非常相似，但是Vector的大小能根据需要动态的变化。

和数组一样，Vector对象的元素也能通过索引访问。

使用Vector类最主要的好处就是在创建对象的时候不必给对象指定大小，它的大小会根据需要动态的变化。

关于该类的更多信息，请参见向量(Vector)

## 栈（Stack）

栈（Stack）实现了一个后进先出（LIFO）的数据结构。

你可以把栈理解为对象的垂直分布的栈，当你添加一个新元素时，就将新元素放在其他元素的顶部。

当你从栈中取元素的时候，就从栈顶取一个元素。换句话说，最后进栈的元素最先被取出。

关于该类的更多信息，请参见栈（Stack）。

## 字典（Dictionary）

字典（Dictionary） 类是一个抽象类，它定义了键映射到值的数据结构。

当你想要通过特定的键而不是整数索引来访问数据的时候，这时候应该使用Dictionary。

由于Dictionary类是抽象类，所以它只提供了键映射到值的数据结构，而没有提供特定的实现。

关于该类的更多信息，请参见字典（ Dictionary）。

## 哈希表（Hashtable）

Hashtable类提供了一种在用户定义键结构的基础上来组织数据的手段。

例如，在地址列表的哈希表中，你可以根据邮政编码作为键来存储和排序数据，而是通过人的名字。

哈希表键的具体含义完全取决于哈希表的使用情景和它包含的数据。

关于该类的更多信息，请参见哈希表（HashTable）。

## 属性（Properties）

Properties 继承于 Hashtable.Properties 类表示了一个持久的属性集.属性列表中每个键及其对应值都是一个字符串。

Properties 类被许多Java类使用。例如，在获取环境变量时它就作为System.getProperties()方法的返回值。

关于该类的更多信息，请参见属性（Properties）。

# Java Enumeration接口

Enumeration接口中定义了一些方法，通过这些方法可以枚举（一次获得一个）对象集合中的元素。

这种传统接口已被迭代器取代，虽然Enumeration 还未被遗弃，但在现代代码中已经被很少使用了。尽管如此，它还是使用在诸如Vector和Properties这些传统类所定义的方法中，除此之外，还用在一些API类，并且在应用程序中也广泛被使用。 下表总结了一些Enumeration声明的方法：

| 方法 | 描述 |
| --- | --- |
| **boolean hasMoreElements( )** | 测试此枚举是否包含更多的元素。 |
| **Object nextElement( )** | 如果此枚举对象至少还有一个可提供的元素，则返回此枚举的下一个元素。 |

## 实例

以下实例演示了Enumeration的使用：

```
import java.util.Vector;
import java.util.Enumeration;

public class EnumerationTester {

   public static void main(String args[]) {
      Enumeration days;
      Vector dayNames = new Vector();
      dayNames.add("Sunday");
      dayNames.add("Monday");
      dayNames.add("Tuesday");
      dayNames.add("Wednesday");
      dayNames.add("Thursday");
      dayNames.add("Friday");
      dayNames.add("Saturday");
      days = dayNames.elements();
      while (days.hasMoreElements()){
         System.out.println(days.nextElement());
      }
   }
}
```

以上实例编译运行结果如下：

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

# Java Bitset类

一个Bitset类创建一种特殊类型的数组来保存位值。BitSet中数组大小会随需要增加。这和位向量（vector of bits）比较类似。

这是一个传统的类，但它在Java 2中被完全重新设计。

BitSet定义了两个构造方法。

第一个构造方法创建一个默认的对象：

```
BitSet()
```

第二个方法允许用户指定初始大小。所有位初始化为0。

```
BitSet(int size)
```

BitSet中实现了Cloneable接口中定义的方法如下表所列：

| 方法 | 描述 |
| --- | --- |
| void and(BitSet bitSet) | 对此目标位 set 和参数位 set 执行逻辑与操作。 |
| void andNot(BitSet bitSet) | 清除此 BitSet 中所有的位，其相应的位在指定的 BitSet 中已设置。 |
| int cardinality( ) | 返回此 BitSet 中设置为 true 的位数。 |
| void clear( ) | 将此 BitSet 中的所有位设置为 false。 |
| void clear(int index) | 将索引指定处的位设置为 false。 |
| void clear(int startIndex, int endIndex) | 将指定的 fromIndex（包括）到指定的 toIndex（不包括）范围内的位设置为 false。 |
| Object clone( ) | 复制此 BitSet，生成一个与之相等的新 BitSet。 |
| boolean equals(Object bitSet) | 将此对象与指定的对象进行比较。 |
| void flip(int index) | 将指定索引处的位设置为其当前值的补码。 |
| void flip(int startIndex, int endIndex) | 将指定的 fromIndex（包括）到指定的 toIndex（不包括）范围内的每个位设置为其当前值的补码。 |
| boolean get(int index) | 返回指定索引处的位值。 |
| BitSet get(int | 返回一个新的 BitSet，它由此 BitSet 中从 |

| | |
|---|---|
| startIndex, int endIndex) | fromIndex（包括）到 toIndex（不包括）范围内的位组成。 |
| int hashCode( ) | 返回此位 set 的哈希码值。 |
| boolean intersects(BitSet bitSet) | 如果指定的 BitSet 中有设置为 true 的位，并且在此 BitSet 中也将其设置为 true，则返回 ture。 |
| boolean isEmpty( ) | 如果此 BitSet 中没有包含任何设置为 true 的位，则返回 ture。 |
| int length( ) | 返回此 BitSet 的"逻辑大小"：BitSet 中最高设置位的索引加 1。 |
| int nextClearBit(int startIndex) | 返回第一个设置为 false 的位的索引，这发生在指定的起始索引或之后的索引上。 |
| int nextSetBit(int startIndex) | 返回第一个设置为 true 的位的索引，这发生在指定的起始索引或之后的索引上。 |
| void or(BitSet bitSet) | 对此位 set 和位 set 参数执行逻辑或操作。 |
| void set(int index) | 将指定索引处的位设置为 true。 |
| void set(int index, boolean v) | 将指定索引处的位设置为指定的值。 |
| void set(int startIndex, int endIndex) | 将指定的 fromIndex（包括）到指定的 toIndex（不包括）范围内的位设置为 true。 |
| void set(int startIndex, int endIndex, boolean v) | 将指定的 fromIndex（包括）到指定的 toIndex（不包括）范围内的位设置为指定的值。 |
| int size( ) | 返回此 BitSet 表示位值时实际使用空间的位数。 |
| String toString( ) | 返回此位 set 的字符串表示形式。 |
| void xor(BitSet bitSet) | 对此位 set 和位 set 参数执行逻辑异或操作。 |

## 实例

下面的程序说明这个数据结构支持的几个方法：

```
import java.util.BitSet;

public class BitSetDemo {

  public static void main(String args[]) {
     BitSet bits1 = new BitSet(16);
     BitSet bits2 = new BitSet(16);

     // set some bits
     for(int i=0; i<16; i++) {
        if((i%2) == 0) bits1.set(i);
        if((i%5) != 0) bits2.set(i);
     }
     System.out.println("Initial pattern in bits1: ");
     System.out.println(bits1);
     System.out.println("\nInitial pattern in bits2: ");
     System.out.println(bits2);

     // AND bits
     bits2.and(bits1);
     System.out.println("\nbits2 AND bits1: ");
     System.out.println(bits2);

     // OR bits
     bits2.or(bits1);
     System.out.println("\nbits2 OR bits1: ");
     System.out.println(bits2);

     // XOR bits
     bits2.xor(bits1);
     System.out.println("\nbits2 XOR bits1: ");
     System.out.println(bits2);
  }
}
```

以上实例编译运行结果如下：

```
Initial pattern in bits1:
{0, 2, 4, 6, 8, 10, 12, 14}

Initial pattern in bits2:
{1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14}

bits2 AND bits1:
{2, 4, 6, 8, 12, 14}

bits2 OR bits1:
{0, 2, 4, 6, 8, 10, 12, 14}

bits2 XOR bits1:
{}
```

# Java Vector 类

Vector类实现了一个动态数组。和ArrayList和相似，但是两者是不同的：

- Vector是同步访问的。
- Vector包含了许多传统的方法，这些方法不属于集合框架。

Vector主要用在事先不知道数组的大小，或者只是需要一个可以改变大小的数组的情况。

Vector类支持4种构造方法。

第一种构造方法创建一个默认的向量，默认大小为10：

```
Vector()
```

第二种构造方法创建指定大小的向量。

```
Vector(int size)
```

第三种构造方法创建指定大小的向量，并且增量用incr指定. 增量表示向量每次增加的元素数目。

```
Vector(int size,int incr)
```

第四中构造方法创建一个包含集合c元素的向量：

```
Vector(Collection c)
```

除了从父类继承的方法外Vector还定义了以下方法：

| 方法 | 描述 |
| --- | --- |
| void add(int index, Object element) | 在此向量的指定位置插入指定的元素。 |
| boolean add(Object o) | 将指定元素添加到此向量的末尾。 |
| boolean addAll(Collection c) | 将指定 Collection 中的所有元素添加到此向量的末尾，按照指定 collection 的迭代器所返回的顺序添加这些元素。 |
| boolean addAll(int index, Collection c) | 在指定位置将指定 Collection 中的所有元素插入到此向量中。 |

| void addElement(Object obj) | 将指定的组件添加到此向量的末尾，将其大小增加 1。 | |
|---|---|---|
| int capacity() | 返回此向量的当前容量。 | |
| void clear() | 从此向量中移除所有元素。 | |
| Object clone() | 返回向量的一个副本。 | |
| boolean contains(Object elem) | 如果此向量包含指定的元素，则返回 true。 | |
| boolean containsAll(Collection c) | 如果此向量包含指定 Collection 中的所有元素，则返回 true。 | |
| void copyInto(Object[] anArray) | 将此向量的组件复制到指定的数组中。 | |
| Object elementAt(int index) | 返回指定索引处的组件。 | |
| Enumeration elements() | 返回此向量的组件的枚举。 | |
| void ensureCapacity(int minCapacity) | 增加此向量的容量（如有必要），以确保其至少能够保存最小容量参数指定的组件数。 | |
| boolean equals(Object o) | 比较指定对象与此向量的相等性。 | |
| Object firstElement() | 返回此向量的第一个组件（位于索引 0) | 处的项）。 |
| Object get(int index) | 返回向量中指定位置的元素。 | |
| int hashCode() | 返回此向量的哈希码值。 | |
| int indexOf(Object elem) | 返回此向量中第一次出现的指定元素的索引，如果此向量不包含该元素，则返回 -1。 | |
| int indexOf(Object elem, int index) | 返回此向量中第一次出现的指定元素的索引，从 index 处正向搜索，如果未找到该元素，则返回 -1。 | |
| void insertElementAt(Object obj, int index) | 将指定对象作为此向量中的组件插入到指定的 index 处。 | |
| boolean isEmpty() | 测试此向量是否不包含组件。 | |
| | | |

| | |
|---|---|
| Object lastElement() | 返回此向量的最后一个组件。 |
| int lastIndexOf(Object elem) | 返回此向量中最后一次出现的指定元素的索引；如果此向量不包含该元素，则返回 -1。 |
| int lastIndexOf(Object elem, int index) | 返回此向量中最后一次出现的指定元素的索引，从 index 处逆向搜索，如果未找到该元素，则返回 -1。 |
| Object remove(int index) | 移除此向量中指定位置的元素。 |
| boolean remove(Object o) | 移除此向量中指定元素的第一个匹配项，如果向量不包含该元素，则元素保持不变。 |
| boolean removeAll(Collection c) | 从此向量中移除包含在指定 Collection 中的所有元素。 |
| void removeAllElements() | 从此向量中移除全部组件，并将其大小设置为零。 |
| boolean removeElement(Object obj) | 从此向量中移除变量的第一个（索引最小的）匹配项。 |
| void removeElementAt(int index) | 删除指定索引处的组件。 |
| protected void removeRange(int fromIndex, int toIndex) | 从此 List 中移除其索引位于 fromIndex（包括）与 toIndex（不包括）之间的所有元素。 |
| boolean retainAll(Collection c) | 在此向量中仅保留包含在指定 Collection 中的元素。 |
| Object set(int index, Object element) | 用指定的元素替换此向量中指定位置处的元素。 |
| void setElementAt(Object obj, int index) | 将此向量指定 index 处的组件设置为指定的对象。 |
| void setSize(int newSize) | 设置此向量的大小。 |
| int size() | 返回此向量中的组件数。 |
| List subList(int fromIndex, int toIndex) | 返回此 List 的部分视图，元素范围为从 fromIndex（包括）到 toIndex（不包括）。 |

| | |
|---|---|
| Object[] toArray() | 返回一个数组，包含此向量中以恰当顺序存放的所有元素。 |
| Object[] toArray(Object[] a) | 返回一个数组，包含此向量中以恰当顺序存放的所有元素；返回数组的运行时类型为指定数组的类型。 |
| String toString() | 返回此向量的字符串表示形式，其中包含每个元素的 String 表示形式。 |
| void trimToSize() | 对此向量的容量进行微调，使其等于向量的当前大小。 |

## 实例

下面的程序说明这个集合所支持的几种方法：

```
import java.util.*;

public class VectorDemo {

   public static void main(String args[]) {
      // initial size is 3, increment is 2
      Vector v = new Vector(3, 2);
      System.out.println("Initial size: " + v.size());
      System.out.println("Initial capacity: " +
      v.capacity());
      v.addElement(new Integer(1));
      v.addElement(new Integer(2));
      v.addElement(new Integer(3));
      v.addElement(new Integer(4));
      System.out.println("Capacity after four additions: " +
          v.capacity());

      v.addElement(new Double(5.45));
      System.out.println("Current capacity: " +
      v.capacity());
      v.addElement(new Double(6.08));
      v.addElement(new Integer(7));
      System.out.println("Current capacity: " +
      v.capacity());
      v.addElement(new Float(9.4));
      v.addElement(new Integer(10));
      System.out.println("Current capacity: " +
      v.capacity());
      v.addElement(new Integer(11));
      v.addElement(new Integer(12));
      System.out.println("First element: " +
         (Integer)v.firstElement());
      System.out.println("Last element: " +
         (Integer)v.lastElement());
      if(v.contains(new Integer(3)))
         System.out.println("Vector contains 3.");
      // enumerate the elements in the vector.
      Enumeration vEnum = v.elements();
      System.out.println("\nElements in vector:");
      while(vEnum.hasMoreElements())
         System.out.print(vEnum.nextElement() + " ");
      System.out.println();
   }
}
```

以上实例编译运行结果如下：

```
Initial size: 0
Initial capacity: 3
Capacity after four additions: 5
Current capacity: 5
Current capacity: 7
Current capacity: 9
First element: 1
Last element: 12
Vector contains 3.

Elements in vector:
1 2 3 4 5.45 6.08 7 9.4 10 11 12
```

# Java Stack 类

栈是Vector的一个子类，它实现了一个标准的后进先出的栈。

堆栈只定义了默认构造函数，用来创建一个空栈。 堆栈除了包括由Vector定义的所有方法，也定义了自己的一些方法。

```
Stack()
```

除了由Vector定义的所有方法，自己也定义了一些方法：

| 方法 | 描述 |
| --- | --- |
| boolean empty() | 测试堆栈是否为空。 |
| Object peek( ) | 查看堆栈顶部的对象，但不从堆栈中移除它。 |
| Object pop( ) | 移除堆栈顶部的对象，并作为此函数的值返回该对象。 |
| Object push(Object element) | 把项压入堆栈顶部。 |
| int search(Object element) | 返回对象在堆栈中的位置，以 1 为基数。 |

## 实例

下面的程序说明这个集合所支持的几种方法

```
import java.util.*;

public class StackDemo {

   static void showpush(Stack st, int a) {
      st.push(new Integer(a));
      System.out.println("push(" + a + ")");
      System.out.println("stack: " + st);
   }

   static void showpop(Stack st) {
      System.out.print("pop -> ");
      Integer a = (Integer) st.pop();
      System.out.println(a);
      System.out.println("stack: " + st);
   }

   public static void main(String args[]) {
      Stack st = new Stack();
      System.out.println("stack: " + st);
      showpush(st, 42);
      showpush(st, 66);
      showpush(st, 99);
      showpop(st);
      showpop(st);
      showpop(st);
      try {
         showpop(st);
      } catch (EmptyStackException e) {
         System.out.println("empty stack");
      }
   }
}
```

以上实例编译运行结果如下：

```
stack: [ ]
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: [ ]
pop -> empty stack
```

# Java Dictionary 类

Dictionary 类是一个抽象类，用来存储键/值对，作用和Map类相似。

给出键和值，你就可以将值存储在Dictionary对象中。一旦该值被存储，就可以通过它的键来获取它。所以和Map一样， Dictionary 也可以作为一个键/值对列表。

Dictionary定义的抽象方法如下表所示：

| 方法 | 描述 |
|---|---|
| **Enumeration elements( )** | 返回此 dictionary 中值的枚举。 |
| **Object get(Object key)** | 返回此 dictionary 中该键所映射到的值。 |
| **boolean isEmpty( )** | 测试此 dictionary 是否不存在从键到值的映射。 |
| **Enumeration keys( )** | 返回此 dictionary 中的键的枚举。 |
| **Object put(Object key, Object value)** | 将指定 key 映射到此 dictionary 中指定 value。 |
| **Object remove(Object key)** | 从此 dictionary 中移除 key （及其相应的 value）。 |
| **int size( )** | 返回此 dictionary 中条目（不同键）的数量。 |

Dictionary类已经过时了。在实际开发中，你可以实现Map接口来获取键/值的存储功能。

# Java Hashtable 接口

Hashtable是原始的java.util的一部分， 是一个Dictionary具体的实现 。

然而，Java 2 重构的Hashtable实现了Map接口，因此，Hashtable现在集成到了集合框架中。它和HashMap类很相似，但是它支持同步。

像HashMap一样，Hashtable在哈希表中存储键/值对。当使用一个哈希表，要指定用作键的对象，以及要链接到该键的值。

然后，该键经过哈希处理，所得到的散列码被用作存储在该表中值的索引。

Hashtable定义了四个构造方法。第一个是默认构造方法：

```
Hashtable()
```

第二个构造函数创建指定大小的哈希表：

```
Hashtable(int size)
```

第三个构造方法创建了一个指定大小的哈希表，并且通过fillRatio指定填充比例。

填充比例必须介于0.0和1.0之间，它决定了哈希表在重新调整大小之前的充满程度：

```
Hashtable(int size,float fillRatio)
```

第四个构造方法创建了一个以M中元素为初始化元素的哈希表。

哈希表的容量被设置为M的两倍。

```
Hashtable(Map m)
```

Hashtable中除了从Map接口中定义的方法外，还定义了以下方法：

| 方法 | 描述 |
|---|---|
| **void clear( )** | 将此哈希表清空，使其不包含任何键。 |
| **Object clone( )** | 创建此哈希表的浅表副本。 |
| **boolean contains(Object value)** | 测试此映射表中是否存在与指定值关联的键。 |
| **boolean containsKey(Object key)** | 测试指定对象是否为此哈希表中的键。 |
| **boolean containsValue(Object value)** | 如果此 Hashtable 将一个或多个键映射到此值，则返回 true。 |
| **Enumeration elements( )** | 返回此哈希表中的值的枚举。 |
| **Object get(Object key)** | 返回指定键所映射到的值，如果此映射不包含此键的映射，则返回 null. 更确切地讲，如果此映射包含满足 (key.equals(k)) 的从键 k 到值 v 的映射，则此方法返回 v；否则，返回 null。 |
| **boolean isEmpty( )** | 测试此哈希表是否没有键映射到值。 |
| **Enumeration keys( )** | 返回此哈希表中的键的枚举。 |
| **Object put(Object key, Object value)** | 将指定 key 映射到此哈希表中的指定 value。 |
| **void rehash( )** | 增加此哈希表的容量并在内部对其进行重组，以便更有效地容纳和访问其元素。 |
| **Object remove(Object key)** | 从哈希表中移除该键及其相应的值。 |
| **int size( )** | 返回此哈希表中的键的数量。 |
| **String toString( )** | 返回此 Hashtable 对象的字符串表示形式，其形式为 ASCII 字符 ", " （逗号加空格）分隔开的、括在括号中的一组条目。 |

## 实例

下面的程序说明这个数据结构支持的几个方法：

```
import java.util.*;

public class HashTableDemo {

   public static void main(String args[]) {
      // Create a hash map
      Hashtable balance = new Hashtable();
      Enumeration names;
      String str;
      double bal;

      balance.put("Zara", new Double(3434.34));
      balance.put("Mahnaz", new Double(123.22));
      balance.put("Ayan", new Double(1378.00));
      balance.put("Daisy", new Double(99.22));
      balance.put("Qadir", new Double(-19.08));

      // Show all balances in hash table.
      names = balance.keys();
      while(names.hasMoreElements()) {
         str = (String) names.nextElement();
         System.out.println(str + ": " +
         balance.get(str));
      }
      System.out.println();
      // Deposit 1,000 into Zara's account
      bal = ((Double)balance.get("Zara")).doubleValue();
      balance.put("Zara", new Double(bal+1000));
      System.out.println("Zara's new balance: " +
      balance.get("Zara"));
   }
}
```

以上实例编译运行结果如下：

```
Qadir: -19.08
Zara: 3434.34
Mahnaz: 123.22
Daisy: 99.22
Ayan: 1378.0

Zara's new balance: 4434.34
```

# Java Properties 接口

Properties 继承于 Hashtable.表示一个持久的属性集.属性列表中每个键及其对应值都是一个字符串。

Properties 类被许多Java类使用。例如，在获取环境变量时它就作为System.getProperties()方法的返回值。

Properties 定义如下实例变量.这个变量持有一个Properties对象相关的默认属性列表。

```
Properties defaults;
```

Properties类定义了两个构造方法. 第一个构造方法没有默认值。

```
Properties()
```

第二个构造方法使用propDefault 作为默认值。两种情况下，属性列表都为空：

```
Properties(Properties propDefault)
```

除了从Hashtable中所定义的方法，Properties定义了以下方法：

| 方法 | 描述 |
| --- | --- |
| **String getProperty(String key)** | 用指定的键在此属性列表中搜索属性。 |
| **String getProperty(String key, String defaultProperty)** | 用指定的键在属性列表中搜索属性。 |
| **void list(PrintStream streamOut)** | 将属性列表输出到指定的输出流。 |
| **void list(PrintWriter streamOut)** | 将属性列表输出到指定的输出流。 |
| **void load(InputStream streamIn) throws IOException** | 从输入流中读取属性列表（键和元素对）。 |
| **Enumeration propertyNames( )** | 按简单的面向行的格式从输入字符流中读取属性列表（键和元素对）。 |
| **Object setProperty(String key, String value)** | 调用 Hashtable 的方法 put。 |
| **void store(OutputStream streamOut, String description)** | 以适合使用 load(InputStream)方法加载到 Properties 表中的格式，将此 Properties 表中的属性列表（键和元素对）写入输出流。 |

## 实例

下面的程序说明这个数据结构支持的几个方法：

```
import java.util.*;

public class PropDemo {

   public static void main(String args[]) {
      Properties capitals = new Properties();
      Set states;
      String str;

      capitals.put("Illinois", "Springfield");
      capitals.put("Missouri", "Jefferson City");
      capitals.put("Washington", "Olympia");
      capitals.put("California", "Sacramento");
      capitals.put("Indiana", "Indianapolis");

      // Show all states and capitals in hashtable.
      states = capitals.keySet(); // get set-view of keys
      Iterator itr = states.iterator();
      while(itr.hasNext()) {
         str = (String) itr.next();
         System.out.println("The capital of " +
            str + " is " + capitals.getProperty(str) + ".");
      }
      System.out.println();

      // look for state not in list -- specify default
      str = capitals.getProperty("Florida", "Not Found");
      System.out.println("The capital of Florida is "
          + str + ".");
   }
}
```

以上实例编译运行结果如下：

```
The capital of Missouri is Jefferson City.
The capital of Illinois is Springfield.
The capital of Indiana is Indianapolis.
The capital of California is Sacramento.
The capital of Washington is Olympia.

The capital of Florida is Not Found.
```

# Java 集合框架

早在Java 2中之前，Java就提供了特设类。比如：Dictionary, Vector, Stack, 和Properties这些类用来存储和操作对象组。

虽然这些类都非常有用，但是它们缺少一个核心的，统一的主题。由于这个原因，使用Vector类的方式和使用Properties类的方式有着很大不同。

集合框架被设计成要满足以下几个目标。

- 该框架必须是高性能的。基本集合（动态数组，链表，树，哈希表）的实现也必须是高效的。
- 该框架允许不同类型的集合，以类似的方式工作，具有高度的互操作性。
- 对一个集合的扩展和适应必须是简单的。

为此，整个集合框架就围绕一组标准接口而设计。你可以直接使用这些接口的标准实现，诸如： LinkedList, HashSet, 和 TreeSet等,除此之外你也可以通过这些接口实现自己的集合。

集合框架是一个用来代表和操纵集合的统一架构。所有的集合框架都包含如下内容：

- 接口：是代表集合的抽象数据类型。接口允许集合独立操纵其代表的细节。在面向对象的语言，接口通常形成一个层次。
- 实现（类）：是集合接口的具体实现。从本质上讲，它们是可重复使用的数据结构。
- 算法：是实现集合接口的对象里的方法执行的一些有用的计算，例如：搜索和排序。这些算法被称为多态，那是因为相同的方法可以在相似的接口上有着不同的实现。

除了集合，该框架也定义了几个Map接口和类。Map里存储的是键/值对。尽管Map不是collections，但是它们完全整合在集合中。

## 集合接口

集合框架定义了一些接口。本节提供了每个接口的概述：

| 接口 | 描述 |
| --- | --- |
| Collection | 接口 允许你使用一组对象，是Collection层次结构的根接口。 |
| List | 接口 继承于**Collection**和一个 List实例存储一个有序集合的元素。 |
| Set | 继承于 **Collection**，是一个不包含重复元素的集合。 |
| SortedSet | 继承于Set保存有序的集合。 |
| Map | 将唯一的键映射到值。 |
| Map.Entry | 描述在一个Map中的一个元素（键/值对）。是一个Map的内部类。 |
| SortedMap | 继承于Map，使Key保持在升序排列。 |
| Enumeration | 这是一个传统的接口和定义的方法，通过它可以枚举（一次获得一个）对象集合中的元素。这个传统接口已被迭代器取代。 |

## 集合类

Java提供了一套实现了Collection接口的标准集合类。其中一些是具体类，这些类可以直接拿来使用，而另外一些是抽象类，提供了接口的部分实现。

标准集合类汇总于下表：

| 类 | 描述 |
| --- | --- |
| **AbstractCollection** | 实现了大部分的集合接口。 |
| **AbstractList** | 继承于AbstractCollection 并且实现了大部分List接口。 |
| **AbstractSequentialList** | 继承于 AbstractList ，提供了对数据元素的链式访问而不是随机访问。 |
| LinkedList | 继承于 AbstractSequentialList，实现了一个链表。 |
| ArrayList | 通过继承AbstractList，实现动态数组。 |
| **AbstractSet** | 继承于AbstractCollection 并且实现了大部分Set接口。 |
| HashSet | 继承了AbstractSet，并且使用一个哈希表。 |
| LinkedHashSet | 具有可预知迭代顺序的 `Set` 接口的哈希表和链接列表实现。 |
| TreeSet | 继承于AbstractSet，使用元素的自然顺序对元素进行排序. |
| **AbstractMap** | 实现了大部分的Map接口。 |
| HashMap | 继承了HashMap，并且使用一个哈希表。 |
| TreeMap | 继承了AbstractMap，并且使用一颗树。 |
| WeakHashMap | 继承AbstractMap类，使用弱密钥的哈希表。 |
| LinkedHashMap | 继承于HashMap，使用元素的自然顺序对元素进行排序. |
| IdentityHashMap | 继承AbstractMap类，比较文档时使用引用相等。 |

在前面的教程中已经讨论通过java.util包中定义的类，如下所示：

| 类 | 描述 |
| --- | --- |
| Vector | Vector类实现了一个动态数组。和ArrayList和相似，但是两者是不同的。 |
| Stack | 栈是Vector的一个子类，它实现了一个标准的后进先出的栈。 |
| Dictionary | Dictionary 类是一个抽象类，用来存储键/值对，作用和Map类相似。 |
| Hashtable | Hashtable是原始的java.util的一部分， 是一个Dictionary具体的实现 。 |
| Properties | Properties 继承于 Hashtable.表示一个持久的属性集.属性列表中每个键及其对应值都是一个字符串。 |
| BitSet | 一个Bitset类创建一种特殊类型的数组来保存位值。BitSet中数组大小会随需要增加。 |

一个Bitset类创建一种特殊类型的数组来保存位值。BitSet中数组大小会随需要增加。

# 集合算法

集合框架定义了几种算法，可用于集合和映射。这些算法被定义为集合类的静态方法。

在尝试比较不兼容的类型时，一些方法能够抛出 ClassCastException异常。当试图修改一个不可修改的集合时，抛出UnsupportedOperationException异常。

集合定义三个静态的变量：EMPTY_SET EMPTY_LIST，EMPTY_MAP的。这些变量都不可改变。

| 算法描述 |
| --- |
| Collection Algorithms 这里是一个列表中的所有算法实现。 |

# 如何使用迭代器

通常情况下，你会希望遍历一个集合中的元素。例如，显示集合中的每个元素。

做到这一点最简单的方法是采用一个迭代器，它是一个对象，实现了Iterator 接口或ListIterator接口。

迭代器，使你能够通过循环来得到或删除集合的元素。ListIterator继承了Iterator，以允许双向遍历列表和修改元素。

这里通过实例列出Iterator和listIterator接口提供的所有方法。

| 迭代器方法描述 |
| --- |
| 使用 Java Iterator |

# 如何使用比较器

TreeSet和TreeMap的按照排序顺序来存储元素. 然而，这是通过比较器来精确定义按照什么样的排序顺序。

这个接口可以让我们以不同的方式来排序一个集合。

| 比较器方法描述 |
| --- |
| 使用 Java Comparator 这里通过实例列出Comparator接口提供的所有方法 |

# 总结

Java集合框架为程序员提供了预先包装的数据结构和算法来操纵他们。

集合是一个对象，可容纳其他对象的引用。集合接口声明对每一种类型的集合可以执行的操作。

集合框架的类和接口均在java.util包中。

# Java 泛型

如果我们只写一个排序方法，就能够对整形数组、字符串数组甚至支持排序的任何类型的数组进行排序，这该多好啊。

Java泛型方法和泛型类支持程序员使用一个方法指定一组相关方法，或者使用一个类指定一组相关的类型。

Java泛型（generics）是JDK 5中引入的一个新特性,泛型提供了编译时类型安全检测机制，该机制允许程序员在编译时检测到非法的类型。

使用Java泛型的概念，我们可以写一个泛型方法来对一个对象数组排序。然后，调用该泛型方法来对整型数组、浮点数数组、字符串数组等进行排序。

## 泛型方法

你可以写一个泛型方法，该方法在调用时可以接收不同类型的参数。根据传递给泛型方法的参数类型，编译器适当地处理每一个方法调用。

下面是定义泛型方法的规则：

- 所有泛型方法声明都有一个类型参数声明部分（由尖括号分隔），该类型参数声明部分在方法返回类型之前（在下面例子中的<E>）。
- 每一个类型参数声明部分包含一个或多个类型参数，参数间用逗号隔开。一个泛型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。
- 类型参数能被用来声明返回值类型，并且能作为泛型方法得到的实际参数类型的占位符。
- 泛型方法方法体的声明和其他方法一样。注意类型参数只能代表引用型类型，不能是原始类型（像int,double,char的等）。

### 实例

下面的例子演示了如何使用泛型方法打印不同字符串的元素：

```
public class GenericMethodTest
{
   // 泛型方法 printArray
   public static < E > void printArray( E[] inputArray )
   {
      // 输出数组元素
         for ( E element : inputArray ){
            System.out.printf( "%s ", element );
         }
         System.out.println();
    }

    public static void main( String args[] )
    {
        // 创建不同类型数组： Integer, Double 和 Character
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        System.out.println( "Array integerArray contains:" );
        printArray( intArray  ); // 传递一个整型数组

        System.out.println( "\nArray doubleArray contains:" );
        printArray( doubleArray ); // 传递一个双精度型数组

        System.out.println( "\nArray characterArray contains:" );
        printArray( charArray ); // 传递一个字符型型数组
    }
}
```

编译以上代码，运行结果如下所示：

```
Array integerArray contains:
1 2 3 4 5 6

Array doubleArray contains:
1.1 2.2 3.3 4.4

Array characterArray contains:
H E L L O
```

有界的类型参数:

可能有时候，你会想限制那些被允许传递到一个类型参数的类型种类范围。例如，一个操作数字的方法可能只希望接受Number或者Number子类的实例。这就是有界类型参数的目的。

要声明一个有界的类型参数，首先列出类型参数的名称，后跟extends关键字，最后紧跟它的上界。

## 实例

下面的例子演示了"extends"如何使用在一般意义上的意思"extends"（类）或者"implements"（接口）。该例子中的泛型方法返回三个可比较对象的最大值。

```
public class MaximumTest
{
   // 比较三个值并返回最大值
   public static <T extends Comparable<T>> T maximum(T x, T y, T z)
   {
      T max = x; // 假设x是初始最大值
      if ( y.compareTo( max ) > 0 ){
         max = y; //y 更大
      }
      if ( z.compareTo( max ) > 0 ){
         max = z; // 现在 z 更大
      }
      return max; // 返回最大对象
   }
   public static void main( String args[] )
   {
      System.out.printf( "Max of %d, %d and %d is %d\n\n",
                   3, 4, 5, maximum( 3, 4, 5 ) );

      System.out.printf( "Maxm of %.1f,%.1f and %.1f is %.1f\n\n",
                   6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );

      System.out.printf( "Max of %s, %s and %s is %s\n","pear",
         "apple", "orange", maximum( "pear", "apple", "orange" ) );
   }
}
```

编译以上代码，运行结果如下所示：

```
Maximum of 3, 4 and 5 is 5

Maximum of 6.6, 8.8 and 7.7 is 8.8

Maximum of pear, apple and orange is pear
```

## 泛型类

泛型类的声明和非泛型类的声明类似，除了在类名后面添加了类型参数声明部分。

和泛型方法一样，泛型类的类型参数声明部分也包含一个或多个类型参数，参数间用逗号隔开。一个泛型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。因为他们接受一个或多个参数，这些类被称为参数化的类或参数化的

类型。

## 实例

如下实例演示了我们如何定义一个泛型类:

```
public class Box<T> {

  private T t;

  public void add(T t) {
    this.t = t;
  }

  public T get() {
    return t;
  }

  public static void main(String[] args) {
     Box<Integer> integerBox = new Box<Integer>();
     Box<String> stringBox = new Box<String>();

     integerBox.add(new Integer(10));
     stringBox.add(new String("Hello World"));

     System.out.printf("Integer Value :%d\n\n", integerBox.get());
     System.out.printf("String Value :%s\n", stringBox.get());
  }
}
```

编译以上代码，运行结果如下所示：

```
Integer Value :10

String Value :Hello World
```

# Java序列化

Java 提供了一种对象序列化的机制，该机制中，一个对象可以被表示为一个字节序列，该字节序列包括该对象的数据、有关对象的类型的信息和存储在对象中数据的类型。

将序列化对象写入文件之后，可以从文件中读取出来，并且对它进行反序列化，也就是说，对象的类型信息、对象的数据，还有对象中的数据类型可以用来在内存中新建对象。

整个过程都是Java虚拟机（JVM）独立的，也就是说，在一个平台上序列化的对象可以在另一个完全不同的平台上反序列化该对象。

类ObjectInputStream 和ObjectOutputStream是高层次的数据流，它们包含序列化和反序列化对象的方法。

ObjectOutputStream 类包含很多写方法来写各种数据类型，但是一个特别的方法例外：

```
public final void writeObject(Object x) throws IOException
```

上面的方法序列化一个对象，并将它发送到输出流。相似的ObjectInputStream 类包含如下反序列化一个对象的方法：

```
public final Object readObject() throws IOException,
                                 ClassNotFoundException
```

该方法从流中取出下一个对象，并将对象反序列化。它的返回值为Object，因此，你需要将它转换成合适的数据类型。

为了演示序列化在Java中是怎样工作的，我将使用之前教程中提到的Employee类，假设我们定义了如下的Employee类，该类实现了Serializable 接口。

```
public class Employee implements java.io.Serializable
{
   public String name;
   public String address;
   public transient int SSN;
   public int number;
   public void mailCheck()
   {
      System.out.println("Mailing a check to " + name
                           + " " + address);
   }
}
```

请注意，一个类的对象要想序列化成功，必须满足两个条件：

该类必须实现 java.io.Serializable 对象。

该类的所有属性必须是可序列化的。如果有一个属性不是可序列化的，则该属性必须注明是短暂的。

如果你想知道一个Java标准类是否是可序列化的，请查看该类的文档。检验一个类的实例是否能序列化十分简答，只需要查看该类有没有实现java.io.Serializable接口。

# 序列化对象

ObjectOutputStream 类用来序列化一个对象，如下的SerializeDemo例子实例化了一个Employee对象，并将该对象序列化到一个文件中。

该程序执行后，就创建了一个名为employee.ser文件。该程序没有任何输出，但是你可以通过代码研读来理解程序的作用。

注意：当序列化一个对象到文件时，按照Java的标准约定是给文件一个.ser扩展名。

```
import java.io.*;

public class SerializeDemo
{
   public static void main(String [] args)
   {
      Employee e = new Employee();
      e.name = "Reyan Ali";
      e.address = "Phokka Kuan, Ambehta Peer";
      e.SSN = 11122333;
      e.number = 101;
      try
      {
         FileOutputStream fileOut =
         new FileOutputStream("/tmp/employee.ser");
         ObjectOutputStream out = new ObjectOutputStream(fileOut);
         out.writeObject(e);
         out.close();
         fileOut.close();
         System.out.printf("Serialized data is saved in /tmp/employ
      }catch(IOException i)
      {
          i.printStackTrace();
      }
   }
}
```

# 反序列化对象

下面的DeserializeDemo程序反序列化在SerializeDemo程序中创建Employee对象。

```
import java.io.*;
public class DeserializeDemo
{
   public static void main(String [] args)
   {
      Employee e = null;
      try
      {
         FileInputStream fileIn = new FileInputStream("/tmp/employe
         ObjectInputStream in = new ObjectInputStream(fileIn);
         e = (Employee) in.readObject();
         in.close();
         fileIn.close();
      }catch(IOException i)
      {
         i.printStackTrace();
         return;
      }catch(ClassNotFoundException c)
      {
         System.out.println("Employee class not found");
         c.printStackTrace();
         return;
      }
      System.out.println("Deserialized Employee...");
      System.out.println("Name: " + e.name);
      System.out.println("Address: " + e.address);
      System.out.println("SSN: " + e.SSN);
      System.out.println("Number: " + e.number);
    }
}
```

以上程序编译运行结果如下所示：

```
Deserialized Employee...
Name: Reyan Ali
Address:Phokka Kuan, Ambehta Peer
SSN: 0
Number:101
```

这里要注意以下要点：

readObject() 方法中的try/catch代码块尝试捕获 ClassNotFoundException异常。对于JVM可以反序列化对象，它必须是能够找到字节码的类。如果JVM在反序列化对象的过程中找不到该类，则抛出一个 ClassNotFoundException异常。

注意，readObject()方法的返回值被转化成Employee引用。

当对象被序列化时，属性SSN的值为111222333，但是因为该属性是短暂的，该值没有被发送到输出流。所以反序列化后Employee对象的SSN属性为0。

# Java 网络编程

网络编程是指编写运行在多个设备（计算机）的程序，这些设备都通过网络连接起来。

java.net包中J2SE的API包含有类和接口，它们提供低层次的通信细节。你可以直接使用这些类和接口，来专注于解决问题，而不用关注通信细节。

java.net包中提供了两种常见的网络协议的支持：

- **TCP**： TCP是传输控制协议的缩写，它保障了两个应用程序之间的可靠通信。通常用于互联网协议，被称TCP / IP。
- **UDP**:UDP是用户数据报协议的缩写，一个无连接的协议。提供了应用程序之间要发送的数据的数据包。

本教程主要讲解以下两个主题。

- **Socket** 编程: 这是使用最广泛的网络概念，它已被解释地非常详细
- **URL** 处理: 这部分会在另外的篇幅里讲，点击这里更详细地了解在Java语言中的URL处理。

# Socket 编程

套接字使用TCP提供了两台计算机之间的通信机制。 客户端程序创建一个套接字，并尝试连接服务器的套接字。

当连接建立时，服务器会创建一个Socket对象。客户端和服务器现在可以通过对Socket对象的写入和读取来进行进行通信。

java.net.Socket类代表一个套接字，并且java.net.ServerSocket类为服务器程序提供了一种来监听客户端，并与他们建立连接的机制。

以下步骤在两台计算机之间使用套接字建立TCP连接时会出现：

- 服务器实例化一个ServerSocket对象，表示通过服务器上的端口通信。
- 服务器调用 ServerSocket类 的accept（）方法，该方法将一直等待，直到客户端连接到服务器上给定的端口。
- 服务器正在等待时，一个客户端实例化一个Socket对象，指定服务器名称和端口号来请求连接。
- Socket类的构造函数试图将客户端连接到指定的服务器和端口号。如果通信被建立，则在客户端创建一个Socket对象能够与服务器进行通信。
- 在服务器端，accept()方法返回服务器上一个新的socket引用，该socket连接到客户端的socket。

连接建立后，通过使用I/O流在进行通信。每一个socket都有一个输出流和一个输入流。客户端的输出流连接到服务器端的输入流，而客户端的输入流连接到服务器端的输出流。

TCP是一个双向的通信协议，因此数据可以通过两个数据流在同一时间发送.以下是一些类提供的一套完整的有用的方法来实现sockets。

## ServerSocket 类的方法

服务器应用程序通过使用java.net.ServerSocket类以获取一个端口,并且侦听客户端请求。

ServerSocket类有四个构造方法：

| 方法 | 描述 |
| --- | --- |
| **public ServerSocket(int port) throws IOException** | 创建绑定到特定端口的服务器套接字。 |
| **public ServerSocket(int port, int backlog) throws IOException** | 利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号。 |
| **public ServerSocket(int port, int backlog, InetAddress address) throws IOException** | 使用指定的端口、侦听 backlog 和要绑定到的本地 IP 地址创建服务器。 |
| **public ServerSocket() throws IOException** | 创建非绑定服务器套接字。 |

创建非绑定服务器套接字。 如果ServerSocket构造方法没有抛出异常，就意味着你的应用程序已经成功绑定到指定的端口，并且侦听客户端请求。

这里有一些ServerSocket类的常用方法：

| 方法 | 描述 |
| --- | --- |
| **public int getLocalPort()** | 返回此套接字在其上侦听的端口。 |
| **public Socket accept() throws IOException** | 侦听并接受到此套接字的连接。 |
| **public void setSoTimeout(int timeout)** | 通过指定超时值启用/禁用 SO_TIMEOUT，以毫秒为单位。 |
| **public void bind(SocketAddress host, int backlog)** | 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。 |

## Socket 类的方法

java.net.Socket类代表客户端和服务器都用来互相沟通的套接字。客户端要获取一个Socket对象通过实例化 ， 而 服务器获得一个Socket对象则通过accept()方法的返回值。

Socket类有五个构造方法.

| 方法 | 描述 |
| --- | --- |
| **public Socket(String host, int port) throws UnknownHostException, IOException.** | 创建一个流套接字并将其连接到指定主机上的指定端口号。 |
| **public Socket(InetAddress host, int port) throws IOException** | 创建一个流套接字并将其连接到指定 IP 地址的指定端口号。 |
| **public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** | 创建一个套接字并将其连接到指定远程主机上的指定远程端口。 |
| **public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** | 创建一个套接字并将其连接到指定远程地址上的指定远程端口。 |
| **public Socket()** | 通过系统默认类型的 SocketImpl 创建未连接套接字 |

当Socket构造方法返回，并没有简单的实例化了一个Socket对象，它实际上会尝试连接到指定的服务器和端口。

下面列出了一些感兴趣的方法，注意客户端和服务器端都有一个Socket对象，所以无论客户端还是服务端都能够调用这些方法。

| 方法 | 描述 |
| --- | --- |
| **public void connect(SocketAddress host, int timeout) throws IOException** | 将此套接字连接到服务器，并指定一个超时值。 |
| **public InetAddress getInetAddress()** | 返回套接字连接的地址。 |
| **public int getPort()** | 返回此套接字连接到的远程端口。 |
| **public int getLocalPort()** | 返回此套接字绑定到的本地端口。 |
| **public SocketAddress getRemoteSocketAddress()** | 返回此套接字连接的端点的地址，如果未连接则返回 null。 |
| **public InputStream getInputStream() throws IOException** | 返回此套接字的输入流。 |
| **public OutputStream getOutputStream() throws IOException** | 返回此套接字的输出流。 |
| **public void close() throws IOException** | 关闭此套接字。 |

## InetAddress 类的方法

这个类表示互联网协议(IP)地址。下面列出了Socket编程时比较有用的方法：

| 方法 | 描述 |
|---|---|
| **static InetAddress getByAddress(byte[] addr)** | 在给定原始 IP 地址的情况下，返回 InetAddress 对象。 |
| **static InetAddress getByAddress(String host, byte[] addr)** | 根据提供的主机名和 IP 地址创建 InetAddress。 |
| **static InetAddress getByName(String host)** | 在给定主机名的情况下确定主机的 IP 地址。 |
| **String getHostAddress()** | 返回 IP 地址字符串（以文本表现形式）。 |
| **String getHostName()** | 获取此 IP 地址的主机名。 |
| **static InetAddress getLocalHost()** | 返回本地主机。 |
| **String toString()** | 将此 IP 地址转换为 String。 |

## Socket 客户端实例

如下的GreetingClient 是一个客户端程序，改程序通过socket连接到服务器并发送一个问候，然后等待一个响应。

```
// 文件名 GreetingClient.java
<pre>
import java.net.*;
import java.io.*;

public class GreetingClient
{
   public static void main(String [] args)
   {
      String serverName = args[0];
      int port = Integer.parseInt(args[1]);
      try
      {
         System.out.println("Connecting to " + serverName
                             + " on port " + port);
         Socket client = new Socket(serverName, port);
         System.out.println("Just connected to "
                      + client.getRemoteSocketAddress());
         OutputStream outToServer = client.getOutputStream();
         DataOutputStream out =
                       new DataOutputStream(outToServer);

         out.writeUTF("Hello from "
                      + client.getLocalSocketAddress());
         InputStream inFromServer = client.getInputStream();
         DataInputStream in =
                        new DataInputStream(inFromServer);
         System.out.println("Server says " + in.readUTF());
         client.close();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

## Socket 服务器实例

如下的GreetingServer 程序是一个服务器端应用程序，改程序使用Socket来监听一个指定的端口。

```
$ java GreetingServer 6066
Waiting for client on port 6066...
```

像下面一样开启客户端：

```
$ java GreetingClient localhost 6066
Connecting to localhost on port 6066
Just connected to localhost/127.0.0.1:6066
Server says Thank you for connecting to /127.0.0.1:6066
Goodbye!
```

# Java 发送邮件

使用Java应用程序发送E-mail十分简单，但是首先你应该在你的机器上安装JavaMail API 和Java Activation Framework (JAF) 。

你可以在 JavaMail (Version 1.2) 下载最新的版本。

你可以再 在JAF (Version 1.1.1)下载最新的版本。

下载并解压这些文件，最上层文件夹你会发现很多的jar文件。你需要将mail.jar和activation.jar 添加到你的CLASSPATH中。

如果你使用第三方邮件服务器如QQ的SMTP服务器，可查看文章底部用户认证完整的实例。

## 发送一封简单的 E-mail

下面是一个发送简单E-mail的例子。假设你的localhost已经连接到网络。

```
// 文件名 SendEmail.java

import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendEmail
{
   public static void main(String [] args)
   {
      // 收件人电子邮箱
      String to = "abcd@gmail.com";

      // 发件人电子邮箱
      String from = "web@gmail.com";

      // 指定发送邮件的主机为 localhost
      String host = "localhost";

      // 获取系统属性
      Properties properties = System.getProperties();

      // 设置邮件服务器
      properties.setProperty("mail.smtp.host", host);

      // 获取默认session对象
      Session session = Session.getDefaultInstance(properties);
```

```
        try{
           // 创建默认的 MimeMessage 对象
           MimeMessage message = new MimeMessage(session);

           // Set From: 头部头字段
           message.setFrom(new InternetAddress(from));

           // Set To: 头部头字段
           message.addRecipient(Message.RecipientType.TO,
                                    new InternetAddress(to));

           // Set Subject: 头部头字段
           message.setSubject("This is the Subject Line!");

           // 设置消息体
           message.setText("This is actual message");

           // 发送消息
           Transport.send(message);
           System.out.println("Sent message successfully....");
        }catch (MessagingException mex) {
           mex.printStackTrace();
        }
     }
  }
```

编译并运行这个程序来发送一封简单的E-mail：

```
$ java SendEmail
Sent message successfully....
```

如果你想发送一封e-mail给多个收件人，那么使用下面的方法来指定多个收件人ID：

```
void addRecipients(Message.RecipientType type,
                   Address[] addresses)
throws MessagingException
```

下面是对于参数的描述：

- **type:**要被设置为TO, CC 或者BCC. 这里CC 代表抄送、BCC 代表秘密抄送y. 举例：*Message.RecipientType.TO*
- **addresses:** 这是email ID的数组。在指定电子邮件ID时，你将需要使用InternetAddress()方法。

# 发送一封HTML E-mail

下面是一个发送HTML E-mail的例子。假设你的localhost已经连接到网络。

和上一个例子很相似，除了我们要使用setContent()方法来通过第二个参数为"text/html"，来设置内容来指定要发送HTML内容。

```
// 文件名 SendHTMLEmail.java

import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendHTMLEmail
{
   public static void main(String [] args)
   {

      // 收件人电子邮箱
      String to = "abcd@gmail.com";

      // 发件人电子邮箱
      String from = "web@gmail.com";

      // 指定发送邮件的主机为 localhost
      String host = "localhost";

      // 获取系统属性
      Properties properties = System.getProperties();

      // 设置邮件服务器
      properties.setProperty("mail.smtp.host", host);

      // 获取默认的 Session 对象。
      Session session = Session.getDefaultInstance(properties);

      try{
         // 创建默认的 MimeMessage 对象。
         MimeMessage message = new MimeMessage(session);

         // Set From: 头部头字段
         message.setFrom(new InternetAddress(from));

         // Set To: 头部头字段
         message.addRecipient(Message.RecipientType.TO,
                                  new InternetAddress(to));

         // Set Subject: 头字段
         message.setSubject("This is the Subject Line!");

         // 发送 HTML 消息, 可以插入html标签
         message.setContent("<h1>This is actual message</h1>",
                            "text/html" );
```

```
            // 发送消息
            Transport.send(message);
            System.out.println("Sent message successfully....");
         }catch (MessagingException mex) {
            mex.printStackTrace();
         }
      }
   }
```

编译并运行此程序来发送HTML e-mail：

```
$ java SendHTMLEmail
Sent message successfully....
```

# 发送带有附件的 E-mail

下面是一个发送带有附件的 E-mail的例子。假设你的localhost已经连接到网络。

```
// 文件名 SendFileEmail.java

import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendFileEmail
{
   public static void main(String [] args)
   {

      // 收件人电子邮箱
      String to = "abcd@gmail.com";

      // 发件人电子邮箱
      String from = "web@gmail.com";

      // 指定发送邮件的主机为 localhost
      String host = "localhost";

      // 获取系统属性
      Properties properties = System.getProperties();

      // 设置邮件服务器
      properties.setProperty("mail.smtp.host", host);

      // 获取默认的 Session 对象。
      Session session = Session.getDefaultInstance(properties);
```

```
      try{
         // 创建默认的 MimeMessage 对象。
         MimeMessage message = new MimeMessage(session);

         // Set From: 头部头字段
         message.setFrom(new InternetAddress(from));

         // Set To: 头部头字段
         message.addRecipient(Message.RecipientType.TO,
                                  new InternetAddress(to));

         // Set Subject: 头字段
         message.setSubject("This is the Subject Line!");

         // 创建消息部分
         BodyPart messageBodyPart = new MimeBodyPart();

         // 消息
         messageBodyPart.setText("This is message body");

         // 创建多重消息
         Multipart multipart = new MimeMultipart();

         // 设置文本消息部分
         multipart.addBodyPart(messageBodyPart);

         // 附件部分
         messageBodyPart = new MimeBodyPart();
         String filename = "file.txt";
         DataSource source = new FileDataSource(filename);
         messageBodyPart.setDataHandler(new DataHandler(source));
         messageBodyPart.setFileName(filename);
         multipart.addBodyPart(messageBodyPart);

         // 发送完整消息
         message.setContent(multipart );

         //   发送消息
         Transport.send(message);
         System.out.println("Sent message successfully....");
      }catch (MessagingException mex) {
         mex.printStackTrace();
      }
   }
}
```

编译并运行你的程序来发送一封带有附件的邮件。

```
$ java SendFileEmail
Sent message successfully....
```

# 用户认证部分

如果需要提供用户名和密码给e-mail服务器来达到用户认证的目的，你可以通过如下设置来完成：

```
props.put("mail.smtp.auth", "true");
props.setProperty("mail.user", "myuser");
props.setProperty("mail.password", "mypwd");
```

e-mail其他的发送机制和上述保持一致。

## 需要用户名密码验证邮件发送实例:

本实例以QQ邮件服务器为例，你需要在登录QQ邮箱后台在"设置"=》账号中开启POP3/SMTP服务，如下图所示：




Java 代码如下：

```
// 需要用户名密码邮件发送实例
//文件名 SendEmail2.java
//本实例以QQ邮箱为例，你需要在qq后台设置

import java.util.Properties;

import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
```

```
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendEmail2
{
   public static void main(String [] args)
   {
      // 收件人电子邮箱
      String to = "xxx@qq.com";

      // 发件人电子邮箱
      String from = "xxx@qq.com";

      // 指定发送邮件的主机为 localhost
      String host = "smtp.qq.com";  //QQ 邮件服务器

      // 获取系统属性
      Properties properties = System.getProperties();

      // 设置邮件服务器
      properties.setProperty("mail.smtp.host", host);

      properties.put("mail.smtp.auth", "true");
      // 获取默认session对象
      Session session = Session.getDefaultInstance(properties,new A
        public PasswordAuthentication getPasswordAuthentication()
        {
         return new PasswordAuthentication("xxx@qq.com", "qq邮箱密码
        }
       });

      try{
         // 创建默认的 MimeMessage 对象
         MimeMessage message = new MimeMessage(session);

         // Set From: 头部头字段
         message.setFrom(new InternetAddress(from));

         // Set To: 头部头字段
         message.addRecipient(Message.RecipientType.TO,
                                  new InternetAddress(to));

         // Set Subject: 头部头字段
         message.setSubject("This is the Subject Line!");

         // 设置消息体
         message.setText("This is actual message");

         // 发送消息
         Transport.send(message);
         System.out.println("Sent message successfully....from w3cs
      }catch (MessagingException mex) {
         mex.printStackTrace();
```

```
        }
    }
}
```

# Java 多线程编程

Java给多线程编程提供了内置的支持。一个多线程程序包含两个或多个能并发运行的部分。程序的每一部分都称作一个线程，并且每个线程定义了一个独立的执行路径。

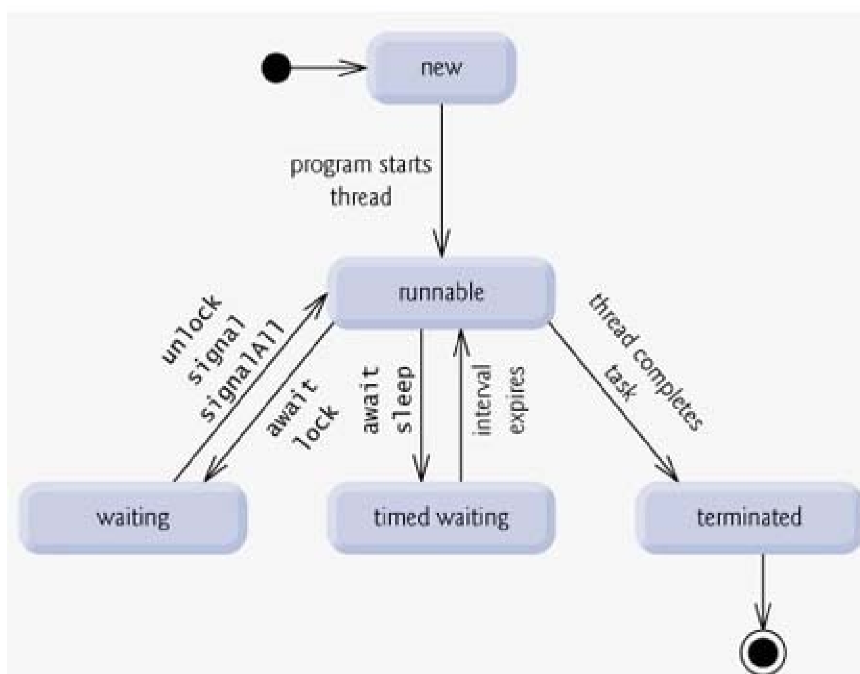
多线程是多任务的一种特别的形式。多线程比多任务需要更小的开销。

这里定义和线程相关的另一个术语：进程：一个进程包括由操作系统分配的内存空间，包含一个或多个线程。一个线程不能独立的存在，它必须是进程的一部分。一个进程一直运行，直到所有的非守候线程都结束运行后才能结束。

多线程能满足程序员编写非常有效率的程序来达到充分利用CPU的目的，因为CPU的空闲时间能够保持在最低限度。

## 一个线程的生命周

线程经过其生命周期的各个阶段。下图显示了一个线程完整的生命周期。




- 新状态: 一个新产生的线程从新状态开始了它的生命周期。它保持这个状态知道程序start这个线程。
- 运行状态:当一个新状态的线程被start以后，线程就变成可运行状态，一个线程在此状态下被认为是开始执行其任务
- 就绪状态:当一个线程等待另外一个线程执行一个任务的时候，该线程就进入就绪状态。当另一个线程给就绪状态的线程发送信号时，该线程才重新切换到运行状态。
- 休眠状态: 由于一个线程的时间片用完了，该线程从运行状态进入休眠状态。当时间间隔到期或者等待的时间发生了，该状态的线程切换到运行状态。

- 终止状态: 一个运行状态的线程完成任务或者其他终止条件发生，该线程就切换到终止状态。

## 线程的优先级

每一个Java线程都有一个优先级，这样有助于操作系统确定线程的调度顺序。Java优先级在MIN_PRIORITY（1）和MAX_PRIORITY（10）之间的范围内。默认情况下，每一个线程都会分配一个优先级NORM_PRIORITY（5）。

具有较高优先级的线程对程序更重要，并且应该在低优先级的线程之前分配处理器时间。然而，线程优先级不能保证线程执行的顺序，而且非常依赖于平台。

## 创建一个线程

Java提供了两种创建线程方法：

- 通过实现Runable接口；
- 通过继承Thread类本身。

## 通过实现**Runnable**接口来创建线程

创建一个线程，最简单的方法是创建一个实现Runnable接口的类。

为了实现Runnable，一个类只需要执行一个方法调用run()，声明如下：

```
publicvoid run()
```

你可以重写该方法，重要的是理解的run()可以调用其他方法，使用其他类，并声明变量，就像主线程一样。

在创建一个实现Runnable接口的类之后，你可以在类中实例化一个线程对象。

Thread定义了几个构造方法，下面的这个是我们经常使用的：

```
Thread(Runnable threadOb,String threadName);
```

这里，threadOb 是一个实现Runnable 接口的类的实例，并且 threadName指定新线程的名字。

新线程创建之后，你调用它的start()方法它才会运行。

```
void start();
```

## 实例

下面是一个创建线程并开始让它执行的实例：

```
// 创建一个新的线程
class NewThread implements Runnable {
   Thread t;
   NewThread() {
      // 创建第二个新线程
      t = new Thread(this, "Demo Thread");
      System.out.println("Child thread: " + t);
      t.start(); // 开始线程
   }

   // 第二个线程入口
   public void run() {
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Child Thread: " + i);
            // 暂停线程
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
         System.out.println("Child interrupted.");
     }
     System.out.println("Exiting child thread.");
   }
}

public class ThreadDemo {
   public static void main(String args[]) {
      new NewThread(); // 创建一个新线程
      try {
         for(int i = 5; i > 0; i--) {
           System.out.println("Main Thread: " + i);
           Thread.sleep(100);
         }
      } catch (InterruptedException e) {
         System.out.println("Main thread interrupted.");
      }
      System.out.println("Main thread exiting.");
   }
}
```

编译以上程序运行结果如下：

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

# 通过继承Thread来创建线程

创建一个线程的第二种方法是创建一个新的类，该类继承Thread类，然后创建一个该类的实例。

继承类必须重写run()方法，该方法是新线程的入口点。它也必须调用start()方法才能执行。

## 实例

```
// 通过继承 Thread 创建线程
class NewThread extends Thread {
   NewThread() {
      // 创建第二个新线程
      super("Demo Thread");
      System.out.println("Child thread: " + this);
      start(); // 开始线程
   }

   // 第二个线程入口
   public void run() {
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Child Thread: " + i);
                            // 让线程休眠一会
            Thread.sleep(50);
         }
      } catch (InterruptedException e) {
         System.out.println("Child interrupted.");
      }
      System.out.println("Exiting child thread.");
   }
}

public class ExtendThread {
   public static void main(String args[]) {
      new NewThread(); // 创建一个新线程
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Main Thread: " + i);
            Thread.sleep(100);
         }
      } catch (InterruptedException e) {
         System.out.println("Main thread interrupted.");
      }
      System.out.println("Main thread exiting.");
   }
}
```

编译以上程序运行结果如下：

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

## Thread 方法

下表列出了Thread类的一些重要方法：

| 方法 | 描述 | |
|---|---|---|
| **public void start()** | 使该线程开始执行；**Java** | 虚拟机调用该线程的 run 方法。 |
| **public void run()** | 如果该线程是使用独立的 Runnable 运行对象构造的，则调用该 Runnable 对象的 run 方法；否则，该方法不执行任何操作并返回。 | |
| **public final void setName(String name)** | 改变线程名称，使之与参数 name 相同。 | |
| **public final void setPriority(int priority)** | 更改线程的优先级。 | |
| **public final void setDaemon(boolean on)** | 将该线程标记为守护线程或用户线程。 | |
| **public final void join(long millisec)** | 等待该线程终止的时间最长为 millis 毫秒。 | |
| **public void interrupt()** | 中断线程。 | |
| **public final boolean isAlive()** | 测试线程是否处于活动状态。 | |

测试线程是否处于活动状态。 上述方法是被Thread对象调用的。下面的方法是Thread类的静态方法。

| 方法 | 描述 |
| --- | --- |
| **public static void yield()** | 暂停当前正在执行的线程对象，并执行其他线程。 |
| **public static void sleep(long millisec)** | 在指定的毫秒数内让当前正在执行的线程休眠（暂停执行），此操作受到系统计时器和调度程序精度和准确性的影响。 |
| **public static boolean holdsLock(Object x)** | 当且仅当当前线程在指定的对象上保持监视器锁时，才返回 true。 |
| **public static Thread currentThread()** | 返回对当前正在执行的线程对象的引用。 |
| **public static void dumpStack()** | 将当前线程的堆栈跟踪打印至标准错误流。 |

## 实例

如下的ThreadClassDemo 程序演示了Thread类的一些方法：

```
// 文件名 : DisplayMessage.java
// 通过实现 Runnable 接口创建线程
public class DisplayMessage implements Runnable
{
   private String message;
   public DisplayMessage(String message)
   {
      this.message = message;
   }
   public void run()
   {
      while(true)
      {
         System.out.println(message);
      }
   }
}
```

```
// 文件名 : GuessANumber.java
// 通过继承 Thread 类创建线程

public class GuessANumber extends Thread
{
   private int number;
   public GuessANumber(int number)
   {
      this.number = number;
   }
   public void run()
   {
      int counter = 0;
      int guess = 0;
      do
      {
          guess = (int) (Math.random() * 100 + 1);
          System.out.println(this.getName()
                       + " guesses " + guess);
          counter++;
      }while(guess != number);
      System.out.println("** Correct! " + this.getName()
                       + " in " + counter + " guesses.**");
   }
}
```

```
// 文件名 : ThreadClassDemo.java
public class ThreadClassDemo
{
   public static void main(String [] args)
   {
      Runnable hello = new DisplayMessage("Hello");
      Thread thread1 = new Thread(hello);
      thread1.setDaemon(true);
      thread1.setName("hello");
      System.out.println("Starting hello thread...");
      thread1.start();

      Runnable bye = new DisplayMessage("Goodbye");
      Thread thread2 = new Thread(hello);
      thread2.setPriority(Thread.MIN_PRIORITY);
      thread2.setDaemon(true);
      System.out.println("Starting goodbye thread...");
      thread2.start();

      System.out.println("Starting thread3...");
      Thread thread3 = new GuessANumber(27);
      thread3.start();
      try
      {
         thread3.join();
      }catch(InterruptedException e)
      {
         System.out.println("Thread interrupted.");
      }
      System.out.println("Starting thread4...");
      Thread thread4 = new GuessANumber(75);

           thread4.start();
      System.out.println("main() is ending...");
   }
}
```

运行结果如下，每一次运行的结果都不一样。

```
Starting hello thread...
Starting goodbye thread...
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Thread-2 guesses 27
Hello
** Correct! Thread-2 in 102 guesses.**
Hello
Starting thread4...
Hello
Hello
..........remaining result produced.
```

## 线程的几个主要概念:

在多线程编程时，你需要了解以下几个概念：

- 线程同步
- 线程间通信
- 线程死锁
- 线程控制：挂起、停止和恢复

## 多线程的使用

有效利用多线程的关键是理解程序是并发执行而不是串行执行的。例如：程序中有两个子系统需要并发执行，这时候就需要利用多线程编程。

通过对多线程的使用，可以编写出非常高效的程序。不过请注意，如果你创建太多的线程，程序执行的效率实际上是降低了，而不是提升了。

请记住，上下文的切换开销也很重要，如果你创建了太多的线程，CPU花费在上下文的切换的时间将多于执行程序的时间！

# Java Applet基础

applet是一种Java程序。它一般运行在支持Java的Web浏览器内。因为它有完整的Java API支持,所以applet是一个全功能的Java应用程序。

如下所示是独立的Java应用程序和applet程序之间重要的不同：

- Java中applet类继承了 java.applet.Applet类
- Applet类没有定义main()，所以一个 Applet程序不会调用main()方法，
- Applets被设计为嵌入在一个HTML页面。
- 当用户浏览包含Applet的HTML页面，Applet的代码就被下载到用户的机器上。
- 要查看一个applet需要JVM。 JVM可以是Web浏览器的一个插件，或一个独立的运行时环境。
- 用户机器上的JVM创建一个applet类的实例，并调用Applet生命周期过程中的各种方法。
- Applets有Web浏览器强制执行的严格的安全规则，applet的安全机制被称为沙箱安全。
- applet需要的其他类可以用Java归档（JAR）文件的形式下载下来。

# Applet的生命周期

Applet类中的四个方法给你提供了一个框架，你可以再该框架上开发小程序：

- **init:** 该方法的目的是为你的applet提供所需的任何初始化。在Applet标记内的param标签被处理后调用该方法。
- **start:** 浏览器调用init方法后，该方法被自动调用。每当用户从其他页面返回到包含Applet的页面时，则调用该方法。
- **stop:**当用户从包含applet的页面移除的时候，该方法自动被调用。因此，可以在相同的applet中反复调用该方法。
- **destroy:** 此方法仅当浏览器正常关闭时调用。因为applets只有在HTML网页上有效，所以你不应该在用户离开包含Applet的页面后遗漏任何资源.
- **paint:** 该方法在start()方法之后立即被调用，或者在applet需要重绘在浏览器的时候调用。paint()方法实际上继承于java.awt。

# "Hello, World" Applet:

下面是一个简单的Applet程序HelloWorldApplet.java:

```
import java.applet.*;
import java.awt.*;

public class HelloWorldApplet extends Applet
{
   public void paint (Graphics g)
   {
      g.drawString ("Hello World", 25, 50);
   }
}
```

这些import语句将以下类导入到我们的applet类中：

```
java.applet.Applet.
java.awt.Graphics.
```

没有这些import语句，Java编译器就识别不了Applet和Graphics类。

# Applet 类

每一个applet都是java.applet.Applet 类的子类，基础的Applet类提供了供衍生类调用的方法,以此来得到浏览器上下文的信息和服务。

这些方法做了如下事情：

- 得到applet的参数
- 得到包含applet的HTML文件的网络位置
- 得到applet类目录的网络位置
- 打印浏览器的状态信息
- 获取一张图片
- 获取一个音频片段
- 播放一个音频片段
- 调整此 applet 的大小

除此之外，Applet类还提供了一个接口，该接口供Viewer或浏览器来获取applet的信息，并且来控制applet的执行。

Viewer可能是：

- 请求applet作者、版本和版权的信息
- 请求applet识别的参数的描述
- 初始化applet
- 销毁applet
- 开始执行applet
- 结束执行applet

Applet类提供了对这些方法的默认实现，这些方法可以在需要的时候重写。

"Hello，World"applet都是按标准编写的。唯一被重写的方法是paint方法。

## Applet的调用

applet是一种Java程序。它一般运行在支持Java的Web浏览器内。因为它有完整的Java API支持,所以applet是一个全功能的Java应用程序。

<applet>标签是在HTML文件中嵌入applet的基础。以下是一个调用"Hello World"applet的例子；

```
<html>
<title>The Hello, World Applet</title>
<hr>
<applet code="HelloWorldApplet.class" width="320" height="120">
If your browser was Java-enabled, a "Hello, World"
message would appear here.
</applet>
<hr>
</html>
```

注意: 你可以参照HTML Applet标签来更多的了解从HTML中调用applet的方法。

<applet>标签的属性指定了要运行的Applet类。Width和height用来指定applet运行面板的初始大小。applet必须使用</applet>标签来关闭。

如果applet接受参数，那么参数的值需要在<param>标签里添加，该标签位于<applet>和</applet>之间。浏览器忽略了applet标签之间的文本和其他标签。

不支持Java的浏览器不能执行<applet>和</applet>。因此，在标签之间显示并且和applet没有关系的任何东西，在不支持的Java的浏览器里是可见的。

Viewer或者浏览器在文档的位置寻找编译过的Java代码，要指定文档的路径，得使用<applet>标签的codebase属性指定。

如下所示：

```
<applet codebase="http://amrood.com/applets"
code="HelloWorldApplet.class" width="320" height="120">
```

如果applet所在一个包中而不是默认包，那么所在的包必须在code属性里指定，例如：

```
<applet code="mypackage.subpackage.TestApplet.class"
           width="320" height="120">
```

# 获得applet参数

下面的例子演示了如何使用一个applet响应来设置文件中指定的参数。该Applet显示了一个黑色棋盘图案和第二种颜色。

第二种颜色和每一列的大小通过文档中的applet的参数指定。

CheckerApplet 在init()方法里得到它的参数。也可以在paint()方法里得到它的参数。然而，在applet开始得到值并保存了设置，而不是每一次刷新的时候都得到值，这样是很方便，并且高效的。

applet viewer或者浏览器在applet每次运行的时候调用init()方法。在加载applet之后，Viewer立即调用init()方法（Applet.init()什么也没做），重写该方法的默认实现，添加一些自定义的初始化代码。

Applet.getParameter()方法通过给出参数名称得到参数值。如果得到的值是数字或者其他非字符数据，那么必须解析为字符串类型。

下例是CheckerApplet.java的梗概：

```
import java.applet.*;
import java.awt.*;
public class CheckerApplet extends Applet
{
   int squareSize = 50;// 初始化默认大小
   public void init () {}
   private void parseSquareSize (String param) {}
   private Color parseColor (String param) {}
   public void paint (Graphics g) {}
}
```

下面是CheckerApplet类的init()方法和私有的parseSquareSize()方法：

```
public void init ()
{
   String squareSizeParam = getParameter ("squareSize");
   parseSquareSize (squareSizeParam);
   String colorParam = getParameter ("color");
   Color fg = parseColor (colorParam);
   setBackground (Color.black);
   setForeground (fg);
}
private void parseSquareSize (String param)
{
   if (param == null) return;
   try {
      squareSize = Integer.parseInt (param);
   }
   catch (Exception e) {
     // 保留默认值
   }
}
```

该applet调用parseSquareSize()，来解析squareSize参数。parseSquareSize()调用了库方法Integer. parseInt()，该方法将一个字符串解析为一个整数，当参数无效的时候，Integer.parseInt()抛出异常。

因此，parseSquareSize()方法也是捕获异常的，并不允许applet接受无效的输入。

Applet调用parseColor()方法将颜色参数解析为一个Color值。parseColor()方法做了一系列字符串的比较，来匹配参数的值和预定义颜色的名字。你需要实现这些方法来使applet工作。

# 指定applet参数

如下的例子是一个HTML文件，其中嵌入了CheckerApplet类。HTML文件通过使用<param>标签的方法给applet指定了两个参数。

```
<html>
<title>Checkerboard Applet</title>
<hr>
<applet code="CheckerApplet.class" width="480" height="320">
<param name="color" value="blue">
<param name="squaresize" value="30">
</applet>
<hr>
</html>
```

**注意:** 参数名字大小写不敏感。

# 应用程序转换成Applet

将图形化的Java应用程序（是指，使用AWT的应用程序和使用java程序启动器启动的程序）转换成嵌入在web页面里的applet是很简单的。

下面是将应用程序转换成applet的几个步骤：

- 编写一个HTML页面，该页面带有能加载applet代码的标签。
- 编写一个JApplet类的子类，将该类设置为public。否则，applet不能被加载。
- 消除应用程序的main()方法。不要为应用程序构造框架窗口，因为你的应用程序要显示在浏览器中。
- 将应用程序中框架窗口的构造方法里的初始化代码移到applet的init()方法中，你不必显示的构造applet对象，浏览器将通过调用init()方法来实例化一个对象。
- 移除对setSize()方法的调用，对于applet来讲，大小已经通过HTML文件里的width和height参数设定好了。
- 移除对 setDefaultCloseOperation()方法的调用。Applet不能被关闭，它随着浏览器的退出而终止。
- 如果应用程序调用了setTitle()方法，消除对该方法的调用。applet不能有标题栏。（当然你可以给通过html的title标签给网页自身命名）
- 不要调用setVisible(true),applet是自动显示的。

# 事件处理

Applet类从Container类继承了许多事件处理方法。Container类定义了几个方法，例如：processKeyEvent()和processMouseEvent()，用来处理特别类型的事件，还有一个捕获所有事件的方法叫做processEvent。

为了响应一个事件，applet必须重写合适的事件处理方法。

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.applet.Applet;
import java.awt.Graphics;

public class ExampleEventHandling extends Applet
                             implements MouseListener {

    StringBuffer strBuffer;

    public void init() {
         addMouseListener(this);
         strBuffer = new StringBuffer();
        addItem("initializing the apple ");
    }

    public void start() {
        addItem("starting the applet ");
    }
```

```
    public void stop() {
        addItem("stopping the applet ");
    }

    public void destroy() {
        addItem("unloading the applet");
    }

    void addItem(String word) {
        System.out.println(word);
        strBuffer.append(word);
        repaint();
    }

    public void paint(Graphics g) {
         //Draw a Rectangle around the applet's display area.
        g.drawRect(0, 0,
                      getWidth() - 1,
                      getHeight() - 1);

         //display the string inside the rectangle.
        g.drawString(strBuffer.toString(), 10, 20);
    }

    public void mouseEntered(MouseEvent event) {
    }
    public void mouseExited(MouseEvent event) {
    }
    public void mousePressed(MouseEvent event) {
    }
    public void mouseReleased(MouseEvent event) {
    }

    public void mouseClicked(MouseEvent event) {
         addItem("mouse clicked! ");
    }
}
```

如下调用该applet：

```
<html>
<title>Event Handling</title>
<hr>
<applet code="ExampleEventHandling.class"
width="300" height="300">
</applet>
<hr>
</html>
```

最开始运行，applet显示 "initializing the applet. Starting the applet."，然后你一点击矩形框，就会显示"mouse clicked" 。

# 显示图片

applet能显示GIF,JPEG,BMP等其他格式的图片。为了在applet中显示图片，你需要使用java.awt.Graphics类的drawImage()方法。

如下实例演示了显示图片的所有步骤：

```
import java.applet.*;
import java.awt.*;
import java.net.*;
public class ImageDemo extends Applet
{
  private Image image;
  private AppletContext context;
  public void init()
  {
      context = this.getAppletContext();
      String imageURL = this.getParameter("image");
      if(imageURL == null)
      {
         imageURL = "java.jpg";
      }
      try
      {
         URL url = new URL(this.getDocumentBase(), imageURL);
         image = context.getImage(url);
      }catch(MalformedURLException e)
      {
         e.printStackTrace();
         // Display in browser status bar
         context.showStatus("Could not load image!");
      }
   }
   public void paint(Graphics g)
   {
      context.showStatus("Displaying image");
      g.drawImage(image, 0, 0, 200, 84, null);
      g.drawString("www.javalicense.com", 35, 100);
   }
}
```

如下调用该applet：

```
<html>
<title>The ImageDemo applet</title>
<hr>
<applet code="ImageDemo.class" width="300" height="200">
<param name="image" value="java.jpg">
</applet>
<hr>
</html>
```

# 播放音频

Applet能通过使用java.applet包中的AudioClip接口播放音频。AudioClip接口定义了三个方法：

- **public void play():** 从一开始播放音频片段一次。
- **public void loop():** 循环播放音频片段
- **public void stop():** 停止播放音频片段

为了得到AudioClip对象，你必须调用Applet类的getAudioClip()方法。无论URL指向的是否是一个真实的音频文件，该方法都会立即返回结果。

直到要播放音频文件时，该文件才会下载下来。

如下实例演示了播放音频的所有步骤：

```
import java.applet.*;
import java.awt.*;
import java.net.*;
public class AudioDemo extends Applet
{
   private AudioClip clip;
   private AppletContext context;
   public void init()
   {
      context = this.getAppletContext();
      String audioURL = this.getParameter("audio");
      if(audioURL == null)
      {
         audioURL = "default.au";
      }
      try
      {
         URL url = new URL(this.getDocumentBase(), audioURL);
         clip = context.getAudioClip(url);
      }catch(MalformedURLException e)
      {
         e.printStackTrace();
         context.showStatus("Could not load audio file!");
      }
   }
   public void start()
   {
      if(clip != null)
      {
         clip.loop();
      }
   }
   public void stop()
   {
      if(clip != null)
      {
         clip.stop();
      }
   }
}
```

如下调用applet：

```
<html>
<title>The ImageDemo applet</title>
<hr>
<applet code="ImageDemo.class" width="0" height="0">
<param name="audio" value="test.wav">
</applet>
<hr>
```

你可以使用你电脑上的test.wav来测试上面的实例。

# Java 文档注释

Java只是三种注释方式。前两种分别是// 和/* */，第三种被称作说明注释，它以/** 开始，以 */结束。

说明注释允许你在程序中嵌入关于程序的信息。你可以使用javadoc工具软件来生成信息，并输出到HTML文件中。

说明注释，是你更加方面的记录你的程序的信息。

## javadoc 标签

javadoc工具软件识别以下标签：

| 标签 | 描述 | 示例 |
|---|---|---|
| @author | 标识一个类的作者 | @author description |
| @deprecated | 指名一个过期的类或成员 | @deprecated description |
| {@docRoot} | 指明当前文档根目录的路径 | Directory Path |
| @exception | 标志一个类抛出的异常 | @exception exception-name explanation |
| {@inheritDoc} | 从直接父类继承的注释 | Inherits a comment from the immediate surperclass. |
| {@link} | 插入一个到另一个主题的链接 | {@link name text} |
| {@linkplain} | 插入一个到另一个主题的链接，但是该链接显示纯文本字体 | Inserts an in-line link to another topic. |
| @param | 说明一个方法的参数 | @param parameter-name explanation |
| @return | 说明返回值类型 | @return explanation |
| @see | 指定一个到另一个主题的链接 | @see anchor |
| @serial | 说明一个序列化属性 | @serial description |
| @serialData | 说明通过writeObject( ) 和 writeExternal( )方法写的数据 | @serialData description |
| @serialField | 说明一个ObjectStreamField组件 | @serialField name type description |
| @since | 标记当引入一个特定的变化时 | @since release |
| @throws | 和 @exception标签一样. | The @throws tag has the same meaning as the @exception tag. |
| {@value} | 显示常量的值，该常量必须是static属性。 | Displays the value of a constant, which must be a static field. |
| @version | 指定类的版本 | @version info |

# 文档注释

在开始的/**之后，第一行或几行是关于类、变量和方法的主要描述.

之后，你可以包含一个或多个何种各样的@标签。每一个@标签必须在一个新行的开始或者在一行的开始紧跟星号(*).

多个相同类型的标签应该放成一组。例如，如果你有三个@see标签，可以将它们一个接一个的放在一起。

下面是一个类的说明注释的示例：

```
/*** This class draws a bar chart.
* @author Zara Ali
* @version 1.2
*/
```

# javadoc输出什么

javadoc工具将你Java程序的源代码作为输入，输出一些包含你程序注释的HTML文件。

每一个类的信息将在独自的HTML文件里。javadoc也可以输出继承的树形结构和索引。

由于javadoc的实现不同，工作也可能不同，你需要检查你的Java开发系统的版本等细节，选择合适的Javadoc版本。

## 实例

下面是一个使用说明注释的简单实例。注意每一个注释都在它描述的项目的前面。

在经过javadoc处理之后，SquareNum类的注释将在SquareNum.html中找到。

```
import java.io.*;

/**
* This class demonstrates documentation comments.
* @author Ayan Amhed
* @version 1.2
*/
public class SquareNum {
   /**
   * This method returns the square of num.
   * This is a multiline description. You can use
   * as many lines as you like.
   * @param num The value to be squared.
   * @return num squared.
   */
   public double square(double num) {
      return num * num;
   }
   /**
   * This method inputs a number from the user.
   * @return The value input as a double.
   * @exception IOException On input error.
   * @see IOException
   */
   public double getNumber() throws IOException {
      InputStreamReader isr = new InputStreamReader(System.in);
      BufferedReader inData = new BufferedReader(isr);
      String str;
      str = inData.readLine();
      return (new Double(str)).doubleValue();
   }
   /**
   * This method demonstrates square().
   * @param args Unused.
   * @return Nothing.
   * @exception IOException On input error.
   * @see IOException
   */
   public static void main(String args[]) throws IOException
   {
      SquareNum ob = new SquareNum();
      double val;
      System.out.println("Enter value to be squared: ");
      val = ob.getNumber();
      val = ob.square(val);
      System.out.println("Squared value is " + val);
   }
}
```

如下，使用javadoc工具处理SquareNum.java文件：

```
$ javadoc SquareNum.java
Loading source file SquareNum.java...
Constructing Javadoc information...
Standard Doclet version 1.5.0_13
Building tree for all the packages and classes...
Generating SquareNum.html...
SquareNum.java:39: warning - @return tag cannot be used\
                      in method with void return type.
Generating package-frame.html...
Generating package-summary.html...
Generating package-tree.html...
Generating constant-values.html...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating help-doc.html...
Generating stylesheet.css...
1 warning
$
```

# Java8教程

## JAVA8有什么新的特性？

JAVA8是JAVA编程语言开发的一大特色版本。它的最初版本发布于2014年3月18日。使用Java8版本，提供Java支持功能编程，新的JavaScript引擎，日期时间操作，新的流API等新API。

## 新功能

- **Lambda** 表达式 - 增加函数处理能力到JAVA。
- 方法引用 - 引用函数由他们名称，而不是直接调用它们。使用函数的参数。
- 默认方法 - 接口有默认的方法实现。
- 新工具 - 新的编译器工具和实用程序被添加，如jdeps找出依赖。
- 数据流**API** - 新数据流的API，以方便数据流处理。
- 日期时间**API** - 改进日期时间API。
- 可选- 强调最佳实践，妥善处理空(null)值。
- **Nashorn , JavaScript**引擎 - 一个基于JAVA引擎执行JavaScript代码。

考虑下面的代码片段。

```
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;
import java.util.Comparator;

public class Java8Tester {

   public static void main(String args[]){
      List<String> names1 = new ArrayList<String>();
      names1.add("Mahesh ");
      names1.add("Suresh ");
      names1.add("Ramesh ");
      names1.add("Naresh ");
      names1.add("Kalpesh ");

      List<String> names2 = new ArrayList<String>();
      names2.add("Mahesh ");
      names2.add("Suresh ");
      names2.add("Ramesh ");
      names2.add("Naresh ");
      names2.add("Kalpesh ");

      Java8Tester tester = new Java8Tester();

      System.out.println("Sort using Java 7 syntax: ");
      tester.sortUsingJava7(names1);
      System.out.println(names1);

      System.out.println("Sort using Java 8 syntax: ");
      tester.sortUsingJava8(names2);
      System.out.println(names2);
   }

   private void sortUsingJava7(List<String> names){
      //sort using java 7
      Collections.sort(names, new Comparator<String>() {
         @Override
         public int compare(String s1, String s2) {
            return s1.compareTo(s2);
         }
      });
   }

   private void sortUsingJava8(List<String> names){
      //sort using java 8
      Collections.sort(names, (s1, s2) ->  s1.compareTo(s2));
   }
}
```

运行程序，看到结果。

```
Sort using Java 7 syntax:
[ Kalpesh Mahesh Naresh Ramesh Suresh ]
Sort using Java 8 syntax:
[ Kalpesh Mahesh Naresh Ramesh Suresh ]
```

在这里，sortUsingJava8（）方法使用排序功能使用一个lambda表达式作为参数，以获得排序条件。

# Java8环境设置 - Java8教程

## Java8环境设置

在开始学习本教程之前，我们需要先Java编程语言设置环境，那么这部分指导如何下载和设置Java在机器上。请按照以下步骤来设置环境。

Java SE是免费提供的链接下载Java。所以，下载时根据您的操作系统版本。

按照说明下载java（注意：选择java8版本）并运行.exe在机器上安装Java。一旦机器上安装了Java，还需要设置环境变量指向正确的安装目录：

## 为Windows 2000/ XP设置路径：

假设你已经安装在 C:Program Filesjavajdk 目录：

- 在“我的电脑”右键单击并选择“属性”。
- 在“高级”选项卡下单击“环境变量”按钮。
- 现在，改变“Path”变量，因此，它也包含了路径Java可执行文件。例如，如果路径当前设置为“C:WINDOWSSYSTEM32”，然后更改路径为“C:WINDOWSSYSTEM32;c:Program Filesjavajdkin”。

## 为Linux，UNIX，Solaris和FreeBSD路径设置：

环境变量PATH应设置为指向已安装的Java二进制文件的位置。

例如，如果使用bash作为shell，那么将下面的行添加到文件 '.bashrc‘ 的末尾 export PATH=/path/to/java:$PATH'

## 流行的Java编辑器：

编写Java程序，需要一个文本编辑器。在市场上可用的IDE。可以考虑下列几种：

- Notepad: 在Windows机器上，可以使用像记事本的任何简单的文本编辑器(推荐本教程)，TextPad。
- Netbeans:是一个Java IDE，它是开源和免费的，可从以下地址下载 http://www.netbeans.org/index.html.
- Eclipse: 也是一个Java IDE开发由Eclipse开源社区，可以从下载 http://www.eclipse.org/.

# Java8 Lambda表达式 - Java8教程

Lambda表达式是在Java8中引入的，并号称是Java8的最大的特点. Lambda表达式有利于函数式编程，简化了开发了很多。

## 语法

lambda表达式的特点，它的语法如下面。

```
parameter -> expression body
```

下面是一个lambda表达式的重要特征。

- 可选类型声明 - 无需声明参数的类型。编译器可以从该参数的值推断。
- 可选圆括号参数 - 无需在括号中声明参数。对于多个参数，括号是必需的。
- 可选大括号 - 表达式主体没有必要使用大括号，如果主体中含有一个单独的语句。
- 可选**return**关键字 - 编译器会自动返回值，如果主体有一个表达式返回的值。花括号是必需的，以表明表达式返回一个值。

## 示例

使用所选择的任何编辑器创建下面的java程序 C:/> JAVA

*Java8Tester.java*

```
public class Java8Tester {
   public static void main(String args[]){
      Java8Tester tester = new Java8Tester();

      //with type declaration
      MathOperation addition = (int a, int b) -> a + b;

      //with out type declaration
      MathOperation subtraction = (a, b) -> a - b;

      //with return statement along with curly braces
      MathOperation multiplication = (int a, int b) -> { return a 
      //without return statement and without curly braces
      MathOperation division = (int a, int b) -> a / b;

      System.out.println("10 + 5 = " + tester.operate(10, 5, addit
      System.out.println("10 - 5 = " + tester.operate(10, 5, subtra
      System.out.println("10 x 5 = " + tester.operate(10, 5, multip
      System.out.println("10 / 5 = " + tester.operate(10, 5, divis

      //with parenthesis
      GreetingService greetService1 = message -> System.out.printl

      //without parenthesis
      GreetingService greetService2 = (message) -> System.out.print

      greetService1.sayMessage("Mahesh");
      greetService2.sayMessage("Suresh");
   }

   interface MathOperation {
      int operation(int a, int b);
   }

   interface GreetingService {
      void sayMessage(String message);
   }

   private int operate(int a, int b, MathOperation mathOperation){
      return mathOperation.operation(a, b);
   }
}
```

## 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的如下：

```
C:\JAVA>java Java8Tester
```

看到结果如下：

```
10 + 5 = 15
10 - 5 = 5
10 x 5 = 50
10 / 5 = 2
Hello Mahesh
Hello Suresh
```

以下是对使用上述例子认为是重要的观点。

- lambda表达式主要用于定义内联执行的功能的接口，即只有一个单一的方法接口。在上面的例子中，我们使用不同类型的lambda表达式定义MathOperation接口的opearation方法。然后，我们定义GreetingService的sayMessage实现。
- Lambda表达式消除匿名类的需求，并给出了一个非常简单但功能强大的函数式编程能力。

# 变量的作用域

在lambda表达式，可以指任何最终的变量或有效的最后一个变量（被分配一次）。如果变量被二次赋值，lambda表达式将抛出编译错误。

## 作用域示例

选择使用任何编辑器创建以下java程序在 C:/> JAVA

*Java8Tester.java*

```
public class Java8Tester {
   final static String salutation = "Hello! ";
   public static void main(String args[]){
      GreetingService greetService1 = message -> System.out.printl
      greetService1.sayMessage("Mahesh");
   }
   interface GreetingService {
      void sayMessage(String message);
   }
}
```

## 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果如下：

```
C:\JAVA>java Java8Tester
```

看看以下结果：

```
Hello! Mahesh
```

# Java8方法引用 - Java8教程

方法引用有助于自己的名字指向方法。方法参考描述使用“::”符号。一种方法参考可以用来指向下列类型的方法。

- 静态方法。
- 实例方法。
- 使用new运算符构造函数(TreeSet::new)

## 方法参考实例

使用所选择的任何编辑器创建下面的java程序C:/> JAVA

*Java8Tester.java*

```
import java.util.List;
import java.util.ArrayList;
public class Java8Tester {

   public static void main(String args[]){

      List names = new ArrayList();
      names.add("Mahesh");
      names.add("Suresh");
      names.add("Ramesh");
      names.add("Naresh");
      names.add("Kalpesh");

      names.forEach(System.out::println);
   }
}
```

在这里，我们通过使用System.out:: println方法为静态方法引用。

## 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果

```
C:\JAVA>java Java8Tester
```

看到结果。

```
Mahesh
Suresh
Ramesh
Naresh
Kalpesh
```

# Java8函数式接口 - Java8教程

函数式接口其中有一个单一的功能，以显示出这些接口。例如，一个可比接口使用单个方法compareTo，并且被用于比较目的。 Java8定义被广泛应用于lambda表达式很多函数形式的接口。以下是在java.util.Function包中定义的功能接口列表。

| S.N. | 接口和说明 |
|---|---|
| 1 | **BiConsumer<T,U>** 表示接收两个输入参数和不返回结果的操作。 |
| 2 | **BiFunction<T,U,R>** 表示接受两个参数，并产生一个结果的函数。 |
| 3 | **BinaryOperator<T>** 表示在相同类型的两个操作数的操作，生产相同类型的操作数的结果。 |
| 4 | **BiPredicate<T,U>** 代表两个参数谓词（布尔值函数）。 |
| 5 | **BooleanSupplier** 代表布尔值结果的提供者。 |
| 6 | **Consumer<T>** 表示接受一个输入参数和不返回结果的操作。 |
| 7 | **DoubleBinaryOperator** 代表在两个double值操作数的运算，并产生一个double值结果。 |
| 8 | **DoubleConsumer** 表示接受一个double值参数，不返回结果的操作。 |
| 9 | **DoubleFunction<R>** 表示接受double值参数，并产生一个结果的函数。 |
| 10 | **DoublePredicate** 代表一个double值参数谓词（布尔值函数）。 |
| 11 | **DoubleSupplier** 表示double值结果的提供者。 |
| 12 | **DoubleToIntFunction** 表示接受double值参数，并产生一个int值结果的函数。 |
| 13 | **DoubleToLongFunction** 代表接受一个double值参数，并产生一个long值结果的函数。 |
| 14 | **DoubleUnaryOperator** 表示上产生一个double值结果的单个double值操作数的操作。 |
| 15 | **Function<T,R>** 表示接受一个参数，并产生一个结果的函数。 |
| 16 | **IntBinaryOperator** 表示对两个int值操作数的运算，并产生一个int值结果。 |
| 17 | **IntConsumer** 表示接受单个int值的参数并没有返回结果的操作。 |
| 18 | **IntFunction<R>** 表示接受一个int值参数，并产生一个结果的函数。 |
| 19 | **IntPredicate** 表示一个整数值参数谓词（布尔值函数）。 |

| | |
|---|---|
| 20 | **IntSupplier** 代表整型值的结果的提供者。 |
| 21 | **IntToDoubleFunction** 表示接受一个int值参数，并产生一个double值结果的功能。 |
| 22 | **IntToLongFunction** 表示接受一个int值参数，并产生一个long值结果的函数。 |
| 23 | **IntUnaryOperator** 表示产生一个int值结果的单个int值操作数的运算。 |
| 24 | **LongBinaryOperator** 表示在两个long值操作数的操作，并产生一个long值结果。 |
| 25 | **LongConsumer** 表示接受一个long值参数和不返回结果的操作。 |
| 26 | **LongFunction<R>** 表示接受long值参数，并产生一个结果的函数。 |
| 27 | **LongPredicate** 代表一个long值参数谓词（布尔值函数）。 |
| 28 | **LongSupplier** 表示long值结果的提供者。 |
| 29 | **LongToDoubleFunction** 表示接受double参数，并产生一个double值结果的函数。 |
| 30 | **LongToIntFunction** 表示接受long值参数，并产生一个int值结果的函数。 |
| 31 | **LongUnaryOperator** 表示上产生一个long值结果单一的long值操作数的操作。 |
| 32 | **ObjDoubleConsumer<T>** 表示接受对象值和double值参数，并且没有返回结果的操作。 |
| 33 | **ObjIntConsumer<T>** 表示接受对象值和整型值参数，并返回没有结果的操作。 |
| 34 | **ObjLongConsumer<T>** 表示接受对象的值和long值的说法，并没有返回结果的操作。 |
| 35 | **Predicate<T>** 代表一个参数谓词（布尔值函数）。 |
| 36 | **Supplier<T>** 表示一个提供者的结果。 |
| 37 | **ToDoubleBiFunction<T,U>** 表示接受两个参数，并产生一个double值结果的功能。 |
| 38 | **ToDoubleFunction<T>** 代表一个产生一个double值结果的功能。 |
| 39 | **ToIntBiFunction<T,U>** 表示接受两个参数，并产生一个int值结果的函数。 |
| 40 | **ToIntFunction<T>** 代表产生一个int值结果的功能。 |
| 41 | **ToLongBiFunction<T,U>** 表示接受两个参数，并产生long值结果的功能。 |
| | |

| 42 | **ToLongFunction<T>** 代表一个产生long值结果的功能。 |
|---|---|
| 43 | **UnaryOperator<T>** 表示上产生相同类型的操作数的结果的单个操作数的操作。 |

# 函数接口例子

谓词 Predicate<T> 接口与方法试验（对象）返回一个布尔值功能接口。此接口意味着一个对象被检测为 true 或 false。

选择使用任何编辑器创建以下java程序在 C:/> JAVA

*Java8Tester.java*

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;

public class Java8Tester {
   public static void main(String args[]){

      List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9)

      // Predicate<Integer> predicate = n -> true
      // n is passed as parameter to test method of Predicate inte
      // test method will always return true no matter what value 
      System.out.println("Print all numbers:");
      //pass n as parameter
      eval(list, n->true);

      // Predicate<Integer> predicate1 = n -> n%2 == 0
      // n is passed as parameter to test method of Predicate inte
      // test method will return true if n%2 comes to be zero
      System.out.println("Print even numbers:");
      eval(list, n-> n%2 == 0 );

      // Predicate<Integer> predicate2 = n -> n > 3
      // n is passed as parameter to test method of Predicate inte
      // test method will return true if n is greater than 3.
      System.out.println("Print numbers greater than 3:");
      eval(list, n-> n > 3 );
   }

   public static void eval(List<Integer> list, Predicate<Integer> p
      for(Integer n: list)  {
         if(predicate.test(n)) {
            System.out.println(n + " ");
         }
      }
   }
}
```

在这里，我们使用通过谓语/Predicate 接口，需要一个单一的输入，并返回 boolean 值。

## 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果

```
C:\JAVA>java Java8Tester
```

看到结果。

```
Print all numbers:
1
2
3
4
5
6
7
8
9
Print even numbers:
2
4
6
8
Print numbers greater than 3:
4
5
6
7
8
9
```

# Java8默认方法 - Java8教程

Java8引入的接口默认方法实现一个新的概念。此功能是为了向后兼容性增加，使旧接口可用于利用JAVA8。 lambda表达式的能力，例如，列表或集合接口不具备forEach方法声明。从而增加了这样的方法只会打破收集框架实现。 Java8引入了默认的方法使列表/Collection接口可以拥有forEach默认方法，并实行类实现这些接口不需要实现相同功能。

## 语法

```
public interface vehicle {
    default void print(){
        System.out.println("I am a vehicle!");
    }
}
```

## 多重默认

在接口默认方法，它们是一类实现两个接口使用相同的默认方法，那么如何解决这种模糊性。考虑下面的情况。

```
public interface vehicle {
    default void print(){
        System.out.println("I am a vehicle!");
    }
}
public interface fourWheeler {
    default void print(){
        System.out.println("I am a four wheeler!");
    }
}
```

第一个解决方案是创建一个自己的方法，并覆盖默认实现。

```
public class car implements vehicle, fourWheeler {
    default void print(){
        System.out.println("I am a four wheeler car vehicle!");
    }
}
```

第二个解决方法是调用使用超指定接口的默认方法。

```
public class car implements vehicle, fourWheeler {
    default void print(){
        vehicle.super.print();
    }
}
```

## 静态默认方法

现在，从Java8起接口也可以有静态辅助方法。

```
public interface vehicle {
    default void print(){
        System.out.println("I am a vehicle!");
    }
    static void blowHorn(){
        System.out.println("Blowing horn!!!");
    }
}
```

## 默认方法示例

使用所选择的任何编辑器创建下面的java程序 **C:/> JAVA**

*Java8Tester.java*

```
public class Java8Tester {
   public static void main(String args[]){
      Vehicle vehicle = new Car();
      vehicle.print();
   }
}

interface Vehicle {
   default void print(){
      System.out.println("I am a vehicle!");
   }
   static void blowHorn(){
      System.out.println("Blowing horn!!!");
   }
}

interface FourWheeler {
   default void print(){
      System.out.println("I am a four wheeler!");
   }
}

class Car implements Vehicle, FourWheeler {
   public void print(){
      Vehicle.super.print();
      FourWheeler.super.print();
      Vehicle.blowHorn();
      System.out.println("I am a car!");
   }
}
```

## 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果

```
C:\JAVA>java Java8Tester
```

看到结果。

```
I am a vehicle!
I am a four wheeler!
Blowing horn!!!
I am a car!
```

# Java8数据流 - Java8教程

流/Stream是在JAVA8中引入的一个抽象，可以处理类似SQL语句声明数据。 例如，考虑下面的SQL语句。

```
SELECT max(salary),employee_id,employee_name FROM Employee
```

上面的SQL表达式会自动返回最大薪水员工的细节，没有对开发者的最终做任何计算。在Java中使用集合框架，开发人员必须使用循环，使检查反复。另一个值得关注的是效率，多核处理器可放心，Java开发人员必须编写的并行代码处理，但是非常容易出错。

为了解决这样的问题，JAVA8引入了流的概念，它允许开发者通过声明处理数据，并可以leverate多核架构，而不需要编写任何特定的代码。

## 什么是数据流？

流代表从支持聚合操作源的序列的对象。以下是数据流的特点。

- 元素序列 - 流提供了一组特定类型的以顺序方式元素。流获取/计算需求的元素。它不存储元素。
- 源- 流使用集合，数组或I/O资源为输入源。
- 聚合操作 - 数据流支持如filter, map, limit, reduced, find, match等聚合操作。
- 管道传输 - 大多数流操作的返回流本身使他们的结果可以被管道传输。这些操作被称为中间操作以及它们的功能是利用输入，处理输入和输出返回到目标。collect()方法是终端操作，这是通常出现在管道传输操作结束标记流的结束。
- 自动迭代 - 流操作内部做了反复对比，其中明确迭代需要集合提供源元素。

## 生成数据流

使用Java8，Collection 接口有两个方法来生成流。

- stream() -返回顺序流考虑集合作为其源。
- parallelStream() - 返回并行数据流考虑集合作为其源。

```
List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd'
List<String> filtered = strings.stream().filter(string -> !string.i
```

# ForEach

数据流提供了新的forEach方法遍历该流中的每个元素。考虑下面的例子打印10个随机数字。

```
Random random = new Random();
random.ints().limit(10).forEach(System.out::println);
```

# map

**map**方法用于映射每个元素对应的结果。考虑下面的例子打印唯一的方形数字。

```
List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);
//get list of unique squares
List<Integer> squaresList = numbers.stream().map( i -> i*i).distinc
```

# filter

**filter**方法用于消除基于标准元素。考虑下面的例子打印空字符串计数(总数)。

```
List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd",
//get count of empty string
int count = strings.stream().filter(string -> string.isEmpty()).cou
```

# limit

**limit** 方法用于减少流的大小。考虑下面的例子打印10个随机数字。

```
Random random = new Random();
random.ints().limit(10).forEach(System.out::println);
```

# sorted

sorted方法用来流排序。考虑下面的例子打印10个随机数字的排序顺序。

```
Random random = new Random();
random.ints().limit(10).sorted().forEach(System.out::println);
```

## 并行处理

parallelStream是流进行并行处理的替代方案。考虑下面的例子打印空字符串计数。

```
List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd'
//get count of empty string
int count = strings.parallelStream().filter(string -> string.isEmpt
```

这是很容易在顺序和并行的流之间进行切换。

## 收集器

收集器是用来处理组合在一个数据流的元素的结果。收集器可用于返回一个列表或一个字符串。

```
List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd",
List<String> filtered = strings.stream().filter(string -> !string.i
System.out.println("Filtered List: " + filtered);
String mergedString = strings.stream().filter(string -> !string.isE
System.out.println("Merged String: " + mergedString);
```

## 统计

使用Java8，统计收集器引入计算所有统计数据时，流处理可以做这些。

```
List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

IntSummaryStatistics stats = integers.stream().mapToInt((x) -> x).s

System.out.println("Highest number in List : " + stats.getMax());
System.out.println("Lowest  number in List : " + stats.getMin());
System.out.println("Sum of all numbers : " + stats.getSum());
System.out.println("Average of all  numbers : " + stats.getAverage(
```

# Stream 例子

选择使用任何编辑器创建以下java程序在C:/> JAVA

*Java8Tester.java*

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.IntSummaryStatistics;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
import java.util.Map;

public class Java8Tester {
   public static void main(String args[]){

      System.out.println("Using Java 7: ");
      // Count empty strings
      List<String> strings = Arrays.asList("abc", "", "bc", "efg",
      System.out.println("List: " +strings);
      long count = getCountEmptyStringUsingJava7(strings);
      System.out.println("Empty Strings: " + count);

      count = getCountLength3UsingJava7(strings);
      System.out.println("Strings of length 3: " + count);

      //Eliminate empty string
      List<String> filtered =  deleteEmptyStringsUsingJava7(strings
      System.out.println("Filtered List: " + filtered);

      //Eliminate empty string and join using comma.
      String mergedString = getMergedStringUsingJava7(strings,", ")
      System.out.println("Merged String: " + mergedString);

      List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

      //get list of square of distinct numbers
      List<Integer> squaresList =  getSquares(numbers);
      System.out.println("Squares List: " + squaresList);

      List<Integer> integers = Arrays.asList(1,2,13,4,15,6,17,8,19)
      System.out.println("List: " +integers);
      System.out.println("Highest number in List : " + getMax(integ
      System.out.println("Lowest number in List : " + getMin(intege
      System.out.println("Sum of all numbers : " + getSum(integers)
      System.out.println("Average of all numbers : " + getAverage(i
      System.out.println("Random Numbers: ");
      //print ten random numbers
      Random random = new Random();
      for(int i=0; i < 10; i++){
         System.out.println(random.nextInt());
      }
      System.out.println("Using Java 8: ");
      System.out.println("List: " +strings);
      count = strings.stream().filter(string->string.isEmpty()).cou
      System.out.println("Empty Strings: " + count);
```

```
      count = strings.stream().filter(string -> string.length() == 
      System.out.println("Strings of length 3: " + count);

      filtered = strings.stream().filter(string -> !string.isEmpty(
      System.out.println("Filtered List: " + filtered);

      mergedString = strings.stream().filter(string -> !string.isEm
      System.out.println("Merged String: " + mergedString);

      squaresList = numbers.stream().map( i -> i*i).distinct().coll
      System.out.println("Squares List: " + squaresList);

      System.out.println("List: " +integers);
      IntSummaryStatistics stats = integers.stream().mapToInt((x) -

      System.out.println("Highest number in List : " + stats.getMax
      System.out.println("Lowest  number in List : " + stats.getMin
      System.out.println("Sum of all numbers : " + stats.getSum());
      System.out.println("Average of all  numbers : " + stats.getAv
      System.out.println("Random Numbers: ");
      random.ints().limit(10).sorted().forEach(System.out::println)

      //parallel processing
      count = strings.parallelStream().filter(string -> string.isEm
      System.out.println("Empty Strings: " + count);
   }

   private static int getCountEmptyStringUsingJava7(List<String> st
      int count = 0;
      for(String string: strings){
         if(string.isEmpty()){
            count++;
         }
      }
      return count;
   }
   private static int getCountLength3UsingJava7(List<String> string
      int count = 0;
      for(String string: strings){
         if(string.length() == 3){
            count++;
         }
      }
      return count;
   }
   private static List<String> deleteEmptyStringsUsingJava7(List<St
      List<String> filteredList = new ArrayList<String>();
      for(String string: strings){
         if(!string.isEmpty()){
            filteredList.add(string);
         }
      }
      return filteredList;
```

```
   }

   private static String getMergedStringUsingJava7(List<String> str
      StringBuilder stringBuilder = new StringBuilder();
      for(String string: strings){
         if(!string.isEmpty()){
            stringBuilder.append(string);
            stringBuilder.append(seperator);
         }
      }
      String mergedString = stringBuilder.toString();
      return  mergedString.substring(0, mergedString.length()-2);
   }

   private static List<Integer> getSquares(List<Integer> numbers){
      List<Integer> squaresList = new ArrayList<Integer>();
      for(Integer number: numbers){
         Integer square = new Integer(number.intValue() * number.ir
         if(!squaresList.contains(square)){
            squaresList.add(square);
         }
      }
      return squaresList;
   }

   private static int getMax(List<Integer> numbers){
      int max = numbers.get(0);
      for(int i=1;i< numbers.size();i++){
         Integer number = numbers.get(i);
         if(number.intValue() > max){
            max = number.intValue();
         }
      }
      return max;
   }

   private static int getMin(List<Integer> numbers){
      int min = numbers.get(0);
      for(int i=1;i< numbers.size();i++){
         Integer number = numbers.get(i);
         if(number.intValue() < min){
            min = number.intValue();
         }
      }
      return min;
   }

   private static int getSum(List<Integer> numbers){
      int sum = numbers.get(0);
      for(int i=1;i< numbers.size();i++){
         sum += numbers.get(i).intValue();
      }
      return sum;
```

```
    }
    private static int getAverage(List<Integer> numbers){
       return getSum(numbers) / numbers.size();
    }
}
```

# 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果

```
C:\JAVA>java Java8Tester
```

看到结果

```
Using Java 7:
List: [abc, , bc, efg, abcd, , jkl]
Empty Strings: 2
Strings of length 3: 3
Filtered List: [abc, bc, efg, abcd, jkl]
Merged String: abc, bc, efg, abcd, jkl
Squares List: [9, 4, 49, 25]
List: [1, 2, 13, 4, 15, 6, 17, 8, 19]
Highest number in List : 19
Lowest number in List : 1
Sum of all numbers : 85
Average of all numbers : 9
Random Numbers:
-1279735475
903418352
-1133928044
-1571118911
628530462
18407523
-881538250
-718932165
270259229
421676854
Using Java 8:
List: [abc, , bc, efg, abcd, , jkl]
Empty Strings: 2
Strings of length 3: 3
Filtered List: [abc, bc, efg, abcd, jkl]
Merged String: abc, bc, efg, abcd, jkl
Squares List: [9, 4, 49, 25]
List: [1, 2, 13, 4, 15, 6, 17, 8, 19]
Highest number in List : 19
Lowest  number in List : 1
Sum of all numbers : 85
Average of all  numbers : 9.444444444444445
Random Numbers:
-1009474951
-551240647
-2484714
181614550
933444268
1227850416
1579250773
1627454872
1683033687
1798939493
Empty Strings: 2
```

# Java8 Optional类 - Java8教程

Optional用于包含非空对象的容器对象。Optional对象，用于表示使用不存在null值。这个类有各种实用的方法，以方便代码来处理为可用或不可用，而不是检查null值。它是Java引入, 是类似于在 Guava 中的 Optional 。

## 类声明

以下是java.util.Optional<T>类的声明：

```
public final class Optional<T>
   extends Object
```

## 类方法

| S.N. | 方法及说明 |
|---|---|
| 1 | **static <T> Optional<T> empty()** 返回一个空的 Optional 实例。 |
| 2 | **boolean equals(Object obj)** 表示某个其他对象是否“等于”此 Optional。 |
| 3 | **Optional<T> filter(Predicate<? super T> predicate)** 如果值存在，并且该值给定的谓词匹配，返回一个可选描述值，否则返回一个空 Optional。 |
| 4 | **<U> Optional<U> flatMap(Function<? super T,Optional<U>> mapper)** 如果值存在，应用提供的可选承载映射功能到它，返回结果，否则返回一个空Optional。 |
| 5 | **T get()** 如果值是出现在这个 Optional 中，返回这个值，否则抛出 NoSuchElementException异常。 |
| 6 | **int hashCode()** 返回当前值，哈希码值（如有）或0（零），如果值不存在。 |
| 7 | **void ifPresent(Consumer<? super T> consumer)** 如果值存在，调用指定的使用方提供值，否则什么都不做。 |
| 8 | **boolean isPresent()** 返回true，如果有一个值存在，否则为false。 |
| 9 | **<U> Optional<U> map(Function<? super T,? extends U> mapper)** 如果值存在，应用提供的映射函数，如果结果非空，返回一个Optional描述结果。 |
| 10 | **static <T> Optional<T> of(T value)** 返回一个Optional具有指定当前非空值。 |
| 11 | **static <T> Optional<T> ofNullable(T value)** 返回一个Optional描述指定的值，如果非空，否则返回一个空的Optional。 |
| 12 | **T orElse(T other)** 返回值（如果存在），否则返回other。 |
| 13 | **T orElseGet(Supplier<? extends T> other)** 如果存在，返回值，否则调用其他并返回调用的结果。 |
| 14 | **<X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier)** 返回所含值，如果存在的话，否则抛出将由提供者创建的一个例外。 |
| 15 | **String toString()** 返回此Optional适合调试一个非空字符串表示。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# Optional 例子

选择使用任何编辑器创建以下java程序在 C:/> JAVA

*Java8Tester.java*

```
import java.util.Optional;

public class Java8Tester {
   public static void main(String args[]){
      Java8Tester java8Tester = new Java8Tester();

      Integer value1 =  null;
      Integer value2 =  new Integer(10);
      //Optional.ofNullable - allows passed parameter to be null.
      Optional<Integer> a = Optional.ofNullable(value1);
      //Optional.of - throws NullPointerException if passed paramet
      Optional<Integer> b = Optional.of(value2);

      System.out.println(java8Tester.sum(a,b));
   }

   public Integer sum(Optional<Integer> a, Optional<Integer> b){
      //Optional.isPresent - checks the value is present or not
      System.out.println("First parameter is present: " + a.isPrese

      System.out.println("Second parameter is present: " + b.isPres

      //Optional.orElse - returns the value if present otherwise re
      //the default value passed.
      Integer value1 = a.orElse(new Integer(0));

      //Optional.get - gets the value, value should be present
      Integer value2 = b.get();

      return value1 + value2;
   }
}
```

## 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果

```
C:\JAVA>java Java8Tester
```

看到结果。

```
First parameter is present: false
Second parameter is present: true
10
```

# Java8 Nashorn JavaScript引擎 - Java8教程

使用Java8，Nashorn大大提高了JavaScript 引擎引入，以取代现有的Nashorn Java脚本引擎。Nashorn提供2至10倍更好的性能，因为它直接编译代码在存储器，并传递到字节码JVM.Nashorn使用invokedynamics函数，在Java7引入以提高性能。

## jjs

对于Nashorn引擎，JAVA8引入了一个新的命令行工具，JJS到控制台执行Java脚本代码。

### 解读js文件

创建并保存sample.js在 **C：> JAVA** 文件夹。

*sample.js*

```
print('Hello World!');
```

打开控制台并使用下面的命令。

```
C:\JAVA>jjs sample.js
```

看到结果

```
Hello World!
```

## JJS在交互模式

打开控制台并使用下面的命令

```
C:\JAVA>jjs
jjs> print("Hello, World!")
Hello, World!
jjs> quit()
>>
```

### 传递参数

打开控制台并使用下面的命令。

```
C:\JAVA> jjs -- a b c
jjs> print('letters: ' +arguments.join(", "))
letters: a, b, c
jjs>
```

# 在JAVA调用JavaScript

使用ScriptEngineManager，JavaScript代码用Java编写可以被调用。

## 示例

选择使用任何编辑器创建以下java程序在 C:/> JAVA

*Java8Tester.java*

```
import javax.script.ScriptEngineManager;
import javax.script.ScriptEngine;
import javax.script.ScriptException;

public class Java8Tester {
   public static void main(String args[]){
      ScriptEngineManager scriptEngineManager = new ScriptEngineMan
      ScriptEngine nashorn = scriptEngineManager.getEngineByName("n
      String name = "Mahesh";

      Integer result = null;
      try {
         nashorn.eval("print('" + name + "')");
         result = (Integer) nashorn.eval("10 + 2");
      }catch(ScriptException e){
         System.out.println("Error executing script: "+ e.getMessag
      }
      System.out.println(result.toString());
   }
}
```

## 验证结果

使用javac编译器编译如下类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果

```
C:\JAVA>java Java8Tester
```

看到结果

```
Mahesh
12
```

# 从JavaScript调用Java

下面的例子将展示如何导入和使用Java类的Java脚本。

创建并保存 **sample.js** 在 **c: > JAVA** 文件夹.

*sample.js*

```
var BigDecimal = Java.type('java.math.BigDecimal');

function calculate(amount, percentage) {
   var result = new BigDecimal(amount).multiply(
      new BigDecimal(percentage)).divide(
         new BigDecimal("100"), 2, BigDecimal.ROUND_HALF_EVEN);
   return result.toPlainString();
}
var result = calculate(568000000000000000023,13.9);
print(result);
```

打开控制台并使用下面的命令。

```
C:\JAVA>jjs sample.js
```

看到结果

```
78952000000000000003.20
```

# Java8 日期时间API - Java8教程

使用Java8，新的日期时间API引入覆盖旧的日期时间API的以下缺点。

- 非线程安全 - java.util.Date不是线程安全的，因此开发者必须在使用日期处理并发性问题。新的日期时间API是不可变的，并且没有setter方法。
- 设计不佳 - 默认的开始日期从1900年，开始每月从1天从0开始，所以没有统一。不直接使用方法操作日期。新的API提供了这样操作实用方法。
- 困难的时区处理 - 开发人员必须编写大量的代码来处理时区的问题。新的API设计开发保持特定领域设计。

JAVA8引入了java.time包 - 下一个新的日期时间API。以下是一些在java.time程序包引入重要的类。

- 本地 - 简化日期时间API，没有时间处理区的复杂性。
- 时区 - 专业的日期时间API来处理各种时区。

| SN | 使用描述方法 |
|---|---|
| 1 | 本地日期时间API LocalDate/本地时间和LocalDateTime类简化时区不需要开发。 |
| 2 | 时区日期时间API 时区日期时间API使用的时区是需要考虑的。 |
| 3 | 计时单位枚举 java.time.temporal.ChronoUnit枚举在Java8添加，以取代旧的API用来代表日，月等整数值 |
| 4 | 周期和持续时间 这些类引入到处理时间的差异。 |
| 5 | 时间调节器 TemporalAdjuster是做数学日期。例如，要获得“本月第二个星期六”或“下周二”。 |
| 6 | 向后兼容性 toInstant()方法被添加到可用于将它们转换到新的日期时间的API原始日期和日历对象。 |

# Java8 Base64 - Java8教程

使用 Java8，Base64终于得到了在Java整合。 Java8现在有内置编码器和解码器的Base64编码。在Java8中，我们可以使用三种类型的Base64编码。

- 简单 - 输出映射设置字符在A-ZA-Z0-9+/。编码器不添加任何换行输出和解码器拒绝在A-Za-z0-9+/以外的任何字符。
- **URL** - 输出映射设置字符在A-Za-z0-9+_。输出URL和文件名安全。
- **MIME** - 输出映射到MIME友好的格式。输出表示在每次不超过76个字符行和使用'\r'后跟一个换行符'\n'回车作为行分隔符。无行隔板的存在是为了使编码的结束输出。

## Nested 类

| S.N. | Nested 类 & 描述 |
| --- | --- |
| 1 | **static class Base64.Decoder** 这个类实现了一个解码器使用的Base64编码方案解码字节的数据，在RFC4648和RFC2045规定。 |
| 2 | **static class Base64.Encoder** 这个类实现一个编码器使用的Base64编码方案编码字节数据在RFC4648和RFC2045规定。 |

## 方法

| S.N. | 方法名称 & 描述 |
|---|---|
| 1 | **static Base64.Decoder getDecoder()** 返回Base64.Decoder解码使用基本型base64编码方案。 |
| 2 | **static Base64.Encoder getEncoder()** 返回Base64.Encoder编码使用的基本型base64编码方案。 |
| 3 | **static Base64.Decoder getMimeDecoder()** 返回Base64.Decoder解码使用MIME类型的base64解码方案。 |
| 4 | **static Base64.Encoder getMimeEncoder()** 返回Base64.Encoder编码使用MIME类型base64编码方案。 |
| 5 | **static Base64.Encoder getMimeEncoder(int lineLength, byte[] lineSeparator)** 返回Base64.Encoder编码使用指定的行长度和线分隔的MIME类型base64编码方案。 |
| 6 | **static Base64.Decoder getUrlDecoder()** 返回Base64.Decoder解码使用URL和文件名安全型base64编码方案。 |
| 7 | **static Base64.Encoder getUrlEncoder()** 返回Base64.Decoder解码使用URL和文件名安全型base64编码方案。... |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

## Base64 例子

选择使用任何编辑器创建以下java程序在 C:/> JAVA

*Java8Tester.java*

```
import java.util.Base64;
import java.util.UUID;
import java.io.UnsupportedEncodingException;

public class HelloWorld {

   public static void main(String args[]){
      try {
         // Encode using basic encoder
         String base64encodedString = Base64.getEncoder().encodeToS
         System.out.println("Base64 Encoded String (Basic) :" + bas

         // Decode
         byte[] base64decodedBytes = Base64.getDecoder().decode(bas
         System.out.println("Original String: "+new String(base64de

         base64encodedString = Base64.getUrlEncoder().encodeToStrin
         System.out.println("Base64 Encoded String (URL) :" + base6

         StringBuilder stringBuilder = new StringBuilder();
         for (int i = 0; i < 10; ++i) {
            stringBuilder.append(UUID.randomUUID().toString());
         }

         byte[] mimeBytes = stringBuilder.toString().getBytes("utf-
         String mimeEncodedString = Base64.getMimeEncoder().encodeT
            System.out.println("Base64 Encoded String (MIME) :"+mir
      }catch(UnsupportedEncodingException e){
         System.out.println("Error :"+e.getMessage());
      }
   }
}
```

# 验证结果

使用javac编译如下编译类

```
C:\JAVA>javac Java8Tester.java
```

现在运行Java8Tester看到的结果

```
C:\JAVA>java Java8Tester
```

看到结果

```
Base64 Encoded String (Basic) :VHV0b3JpYWxzUG9pbnQ/amF2YTg=
Original String: YiiBai?java8
Base64 Encoded String (URL) :VHV0b3JpYWxzUG9pbnQ_amF2YTg=
Base64 Encoded String (MIME) :ZWJjY2YzZWUtYmUwZC00Yjg1LTlkYjUtNWUyN
4ZGQ4ZjE1NGJmMjEtNTdkNi00YzM1LTg4
MzYtNDZlYzNhZDM2NTdkZmQzY2RiNzMtMTU1OC00ZjBmLWFmZGQtM2YyZWU3MDYzZjC
WQ0
ODctZWEyZS00YzM2LWEyZmUtOGVkMmNjMGNmZGM3MTg5YWUyZGQtMzg4MS00M2NkLWI
jNh
Zjk2OGIxZDU2YzkzODZlYTUtNjljNC00ZmIyLTkzYTQtMzVlOTFlNjdlY2E0MDcwNWE
mE4
Yy00OTljLTg2NmItMjE3ZTZmMmIyY2NiNzI2MjAwZWQtMjI0NC00YzJhLWJiMGItOTc
zIx
NGFkY2QyZmVhODItNmUyOS00MWNjLWFlODItNzdmNzRhYmQ4NGU5ZGQ3ZjY3NzktZjg
zlk
LTlmNDgtOTNlNTIwYzIzZDcy
```

# java实例教程




汇集可以使用的Java编程实例。 Java运行于各种平台，如Windows，Mac OS和各种版本的UNIX。

这些例子对于项目开发，论文和学习是非常有用的。

## 读者

此参考教程是为初学者，帮助他们了解基本的Java编程语言和先进设计理念。

## 前提条件

在开始做练习以及在此引用给定的例子，假设读者已经知道什么是Java编程和概念。

# 编译/执行Java程序

```
public class MyFirstJavaProgram {

    public static void main(String []args) {
       System.out.println("Hello World");
    }
}
```

## 1 - JAVA面试试题

通过最期望和重要的面试问题来增强自己的信心。

Java 面试题

# Java环境 - java实例教程

学习如何使用Java编程的编程环境。下面是最常用的例子：

1. 如何编译一个java文件?
2. 如何运行一个类文件?
3. 如何调试一个java文件?
4. How to set classpath?
5. 如何查看当前classpath?
6. 如何设置类文件的目标?
7. 如何运行编译后的class文件?
8. 如何检查Java运行在系统上的版本?
9. 如何设置classpath，当类文件均为.jar文件?

# Java String/字符串操作实例 - java实例教程

了解如何在Java编程中的字符串使用。下面是最常用的例子：

1. 如何比较字符串?
2. 如何搜索一个子串的最后一次数出现?
3. 如何从一个字符串中删除特定的字符?
4. 如何通过一个字符串替换字符串中的另一个子串?
5. 如何逆转一个字符串?
6. 如何搜索一个字符串中的单词?
7. 如何将一个字符串分割成若干子字符串 ?
8. 如何将字符串全部转换成大写?
9. 如何在一个字符串匹配区域?
10. 如何比较两个字符串的性能?
11. 如何优化字符串创建?
12. 如何格式化字符串?
13. 如何连接两个字符串?
14. 如何获得字符串的unicode?
15. 如何将字符串缓冲区?

# Java Arrays/数组实例 - java实例教程

了解如何在Java编程中使用数组。下面是最常用的例子：

1. 如何对数组进行排序和搜索里面的元素？
2. 如何排序的数组，并插入元素？
3. 如何确定定义一个二维数组？
4. 如何逆转数组？
5. 如何写一个字符串数组到输出控制台？
6. 如何搜索数组的最小和最大元素？
7. How to merge two arrays?
8. 如何填写（初始化一次）的数组？
9. 如何初始化后延长/扩大一个数组？
10. 如何对数组进行排序和搜索里面的元素？
11. 如何删除数组中的一个元素？
12. 如何从一个数组中删除另一个数组？
13. 如何找到从数组中公共元素？
14. 如何在Array中查找一个对象或一个字符串？
15. 如何检查是否两个数组是否相等？
16. 如何比较两个数组？

# Java日期、时间Date/Time - java实例教程

了解如何在Java编程中使用数据和时间。下面是最常用的例子：

1. 如何在AM-PM格式化时间?
2. 如何在（MMM）格式显示月份名称？
3. 如何显示小时和分钟？
4. 如何显示当前日期和时间?
5. 如何显示24小时制时间格式?
6. 如何在MMMM格式格式化时间 ？
7. 如何格式化秒?
8. 如何在short格式显示的月份名称?
9. 如何显示工作日的名称?
10. 如何添加时间日期?
11. 如何以不同的国家的格式显示时间？
12. 如何在不同的语言显示时间？
13. 如何滚动小时和月份？
14. 如何找到本年度的星期？
15. 如何以不同的格式显示日期？

# Java方法实例 - java实例教程

学习如何使用Java编程的方法。下面是最常用的例子：

1. 如何重载的方法呢？
2. 如何使用方法重载用于打印不同类型的数组？
3. 如何使用方法求解汉诺塔问题？
4. 如何使用方法计算Fibonacci数列？
5. 如何使用方法，用于计算数阶乘？
6. 如何使用方法重载继承的子类？
7. 如何用instanceof关键字来显示Object类？
8. 如何使用break跳出循环的方法？
9. 如何使用continue在方法中？in a method?
10. 如何在方法中使用标签？
11. 如何使用枚举和switch语句？
12. 如何使枚举构造函数，实例变量和方法？
13. 如何使用和foreach循环对数组的值打印？
14. 如何使一个方法把变量lentgth参数作为输入？
15. 如何方法重载处理使用可变参数作为输入？

# Java文件操作实例 - java实例教程

了解如何在Java编程中使用文件。下面是最常用的例子：

1. Java如何比较两个文件的路径？
2. Java如何创建一个新的文件？
3. Java如何得到一个文件的最后修改日期？
4. Java如何在指定目录下创建一个文件？
5. Java如何检查一个文件存在与否？
6. Java如何使一个文件设置为只读？
7. Java如何重命名一个文件？
8. Java如何得到一个文件的大小（字节）？
9. Java如何改变一个文件的最后修改时间？
10. Java如何创建一个临时文件？
11. Java如何追加一个字符串到现有文件？
12. Java如何将一个文件复制到另一个文件？
13. Java如何删除一个文件？
14. Java如何读取一个文件？
15. Java如何写入一个文件？

# Java目录操作实例 - java实例教程

了解Java编程中如何访问目录。下面是最常用的例子：

1. Java如何递归创建目录？
2. Java如何删除一个目录？
3. Java如何获得一个目录为空或不是为空？
4. Java如何获得一个目录是否为隐藏？
5. Java如何打印目录层次结构？
6. Java如何打印目录的最后修改时间？
7. Java如何得到一个文件的父目录？
8. Java如何搜索一个目录内的所有文件？
9. Java如何获得一个目录的大小？
10. Java如何遍历一个目录？
11. Java如何找到当前工作目录？
12. Java如何在系统中显示的根目录？
13. Java如何搜索目录中的文件？
14. Java如何显示所有目录中的文件？
15. Java如何显示在一个目录下的所有目录？

# Java Exception/异常实例 - java实例教程

了解如何在Java编程中使用异常。下面是最常用的例子：

1. Java如何使用finally块用于捕获异常？
2. Java如何使用处理异常层次结构？
3. Java如何使用处理异常的方法？
4. Java如何使用处理运行时异常？
5. Java如何使用处理空栈异常？
6. Java如何使用catch来处理异常？
7. Java如何使用catch来处理异常链？
8. Java如何使用重载方法处理该异常？
9. Java如何处理已检查异常？
10. Java如何传递参数而抛出检查型异常？
11. Java如何处理多个异常（如除以0）？
12. Java如何处理多个异常（数组越界）？
13. Java如何打印异常堆栈？
14. Java如何使用线程异常？
15. Java如何创建用户定义的异常？

# Java数据结构实例 - java实例教程

了解如何在Java编程中使用数据结构。下面是最常用的例子：

1. Java如何打印n个数的总和？
2. Java如何获得链表的第一和最后一个元素？
3. Java如何在一个链表第一个和最后一个位置添加一个元素？
4. Java如何将中缀表达式转换为后缀表达式？
5. Java如何实现队列Queue？
6. Java如何使用堆栈来反转一个字符串？
7. Java如何搜索一个链表里面的元素？
8. Java如何实现栈？
9. Java如何换一个向量的两个元素？
10. Java如何更新一个链表？
11. Java如何从一个向量得到最大的元素？
12. Java如何在向量执行二进制搜索？
13. Java如何获得LinkedList的一个元素？
14. Java如何从链表中删除多个元素？

# Java集合实例 - java实例教程

了解如何在Java编程中使用集合。下面是最常用的例子：

1. Java如何将一个数组转换成一个集合？
2. Java如何比较集合中元素呢？
3. Java如何将集合转换成一个数组？
4. Java如何打印一个集合？
5. Java如何使一个集合只读？
6. Java如何从集合中删除一个特定的元素？
7. Java如何逆转一个集合？
8. Java如何打乱一个集合中的元素？
9. Java如何获得一个集合的大小？
10. Java如何循环HashMap中的元素？
11. Java如何使用不同类型的集合？
12. Java如何使用枚举来显示哈希表的内容是什么？
13. 如何从Java哈希表获得Set视图的键？
14. Java如何从一个列表List中找到最小和最大值？
15. Java如何找出一个列表的子列表？
16. Java如何替换列表中的一个元素？
17. Java如何旋转列表元素？

# Java网络编程实例 - java实例教程

学习如何在Java编程中修改网络。下面是最常用的例子：

1. Java如何将主机名改为其特定的IP地址？
2. Java如何获得与Web服务器连接？
3. Java如何检查一个文件服务器有没有被修改？
4. Java如何建立一个多线程的服务器？
5. Java如何从服务器获取文件的大小？
6. Java如何使显示信息给一个客户机的套接字？
7. Java如何使一个srever允许连接到套接字6123？
8. Java如何获得一个URL的部分？
9. Java如何获得URL连接的日期？
10. Java如何读取和下载一个网页？
11. Java如何从IP地址找到主机名？
12. Java如何确定IP地址和本地计算机的主机名？
13. Java如何检查端口是否被使用？
14. Java如何找到一个系统的代理设置？
15. Java如何在一个特定的端口创建一个socket？

# Java Applet实例 - java实例教程

了解如何在Java编程中使用Applets。下面是最常用的例子：

1. 如何创建一个基本的小程序Applet？
2. 如何使用Applet的横幅？
3. 如何使用Applet来显示时钟？
4. 如何使用Applet的来创建不同的形状？
5. 如何使用Applet填充颜色形状？
6. 如何使用Applet跳转链接？
7. 如何在Applet创建一个事件监听器？
8. 如何使用Applet来显示图像？
9. 如何使用Applet在一个新的窗口中打开链接？
10. 如何使用Applet来播放声音？
11. 如何使用Applet读取一个文件？
12. 如何使用Applet写入一个文件？
13. 如何在JAVA中使用Swing应用程序applet？

# Java简单的图形用户界面-GUI - java实例教程

了解如何在Java编程中使用简单的GUI。下面是最常用的例子：

1. Java如何显示不同字体的文字？
2. Java如何使用GUI来画一条线？
3. Java如何在新的框架中显示的消息？
4. Java如何使用GUI来绘制一个多边形？
5. Java如何在一个矩形显示字符串？
6. Java如何使用GUI来显示不同的形状？
7. Java如何使用GUI绘制实心矩形？
8. Java如何创建一个透明的光标？
9. Java如何检查抗锯齿功能启用与否？
10. Java如何在框架中显示的颜色？
11. Java如何使用显示框架饼图？
12. Java如何使用GUI来绘制文字？

# Java JDBC实例 - java实例教程

了解如何在Java编程中使用JDBC。下面是最常用的例子：

1. JDBC如何与数据库建立连接？
2. JDBC如何创建，编辑和使用Java的alter table？
3. JDBC如何显示数据库表的内容是什么？
4. JDBC如何更新，编辑和删除一行数据？
5. JDBC如何用java命令在数据库中搜索？
6. JDBC如何用java命令排序列的元素呢？
7. JDBC如何从多个表中将数据组合？
8. JDBC如何在Java中使用commit语句？
9. JDBC如何在Java编写的语句？
10. JDBC如何设置和回滚到一个保存点？
11. JDBC如何用java SQL语句执行批处理？
12. JDBC如何在java使用不同的行方法？
13. JDBC如何在java中使用不同的列方法列？

# Java正则表达式实例 - java实例教程

了解如何在Java编程中使用正则表达式。下面是最常用的例子：

1. 如何重置一个正则表达式的模式？
2. 如何在正则表达式匹配重复的单词？
3. 如何找到一个单词的出现次数？
4. 如何知道一个字符串中特定单词的最后一个索引？
5. 如何打印从一个文件中所有给定模式匹配的字符串？
6. 如何去除空格？
7. 如何在列表中特定的模式相匹配电话号码？
8. 如何计算一个字符串的一组词？
9. 如何在一个字符串搜索一个特定的词？
10. 如何拆分一个正则表达式？
11. 怎么算替换第一次出现的字符串？
12. 如何检查日期是否为正确的格式？
13. 如何验证电子邮件地址的格式？
14. 如何更换一个字符串的所有匹配？
15. 如何让每个单词的第一个字符转为大写？

# JavaFX教程

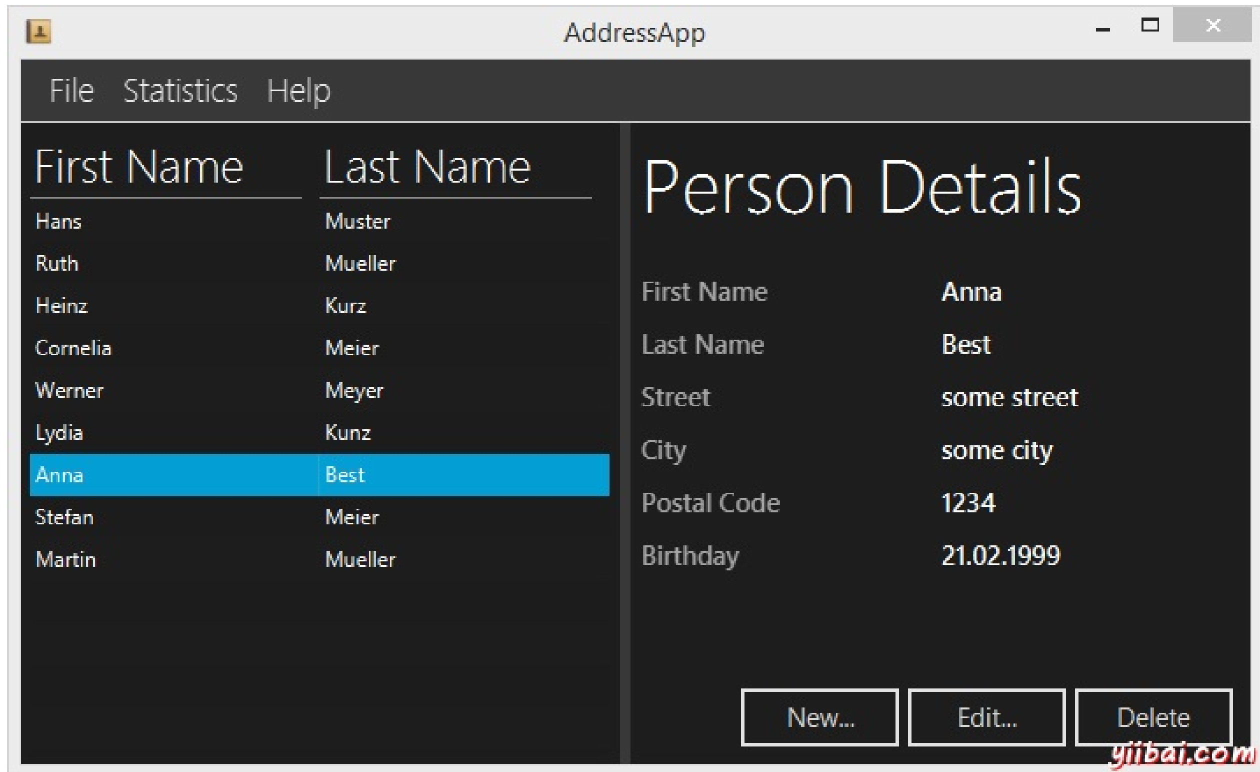
JavaFX 编程语言可用于开发互联网应用程序(RIA）。JavaFX技术主要应用于创建Rich Internet applications（RIAs）。当前的JavaFX包括JavaFX脚本和JavaFX Mobile（一种运营于行动装置的操作系统），今后JavaFX将包括更多的产品。JavaFX Script编程语言（以下称为JavaFX）是一种声明性的、静态类型脚本语言。

JavaFX技术有着良好的前景，包括可以直接调用Java API的能力。因为 JavaFX Script是静态类型，它同样具有结构化代码、重用性和封装性，如包、类、继承和单独编译和发布单元，这些特性使得使用JavaFX技术创建和管理大型程序变为可能。

JavaFX的开发者很赞同让用户发布自己的应用，并坚信开源和社区的力量。但当前的JavaFX版本是在评估授权下发布的，因此并不能够被重新发布。你当然能够发布自己编写的基于JavaFX的应用，但不能够和JavaFX一起捆绑发布，并需要告之使用者：需要到openjfx项目站点下载JavaFX二进制代码库。并且，由于正处于早期的JavaFX代码还处在评估授权下，因此它不能用于商业用途。当Sun完成了JavaFX的商业化版本开发后，相信这将会得到改变。

# JavaFX是什么？ - JavaFX教程

这个教程将指导您设计，编写并部署一个联系人应用程序。应用程序最后将会是这个样子：




## 你将学到什么？

- 创建并启动一个 JavaFX 项目。
- 使用 Scene Builder 设计 UI 。
- 构造一个 模型 - 视图 - 控制器 (MVC) 模式的应用程序。
- 使用 ObservableLists 来自动更新用户界面。
- 使用 TableView 来响应列表中的选择。
- 创建一个 edit persons 的自定义弹出式对话框。
- 验证用户输入。
- 使用 CSS 样式化一个 JavaFX 应用程序。
- 使用 XML 保存数据。
- 在用户配置中保存最后一次打开文件的路径。
- 创建 JavaFX 的统计图表。
- 部署一个 JavaFX 到本机软件包。

这是相当多的！ 所以，当你学习完这个教程后，你应该准备好使用 JavaFX 构建复杂的应用程序。

## 如何使用这个教程？

这有两种使用本教程两种方法：

- 最大化学习的通道： 从头开始创建自己的 JavaFX 项目。
- 快速通道： 导入教程部分的源代码到你的 IDE(它是一个 Eclipse 项目，但是你可以稍作修改后使用其它的 IDE 例如 NetBeans 这样的)。然后再通过教程来理解代码。

现在，我希望你觉得有趣！开始 第一部分：Scene Builder.

# JavaFX - Scene Builder - JavaFX教程

## 第一部分的主题

- 开始了解 JavaFX 。
- 创建并运行一个 JavaFX 项目。
- 使用 Scene Builder 来设计用户界面。
- 使用 模型 - 视图 - 控制器(MVC)模式 构造基础的应用。

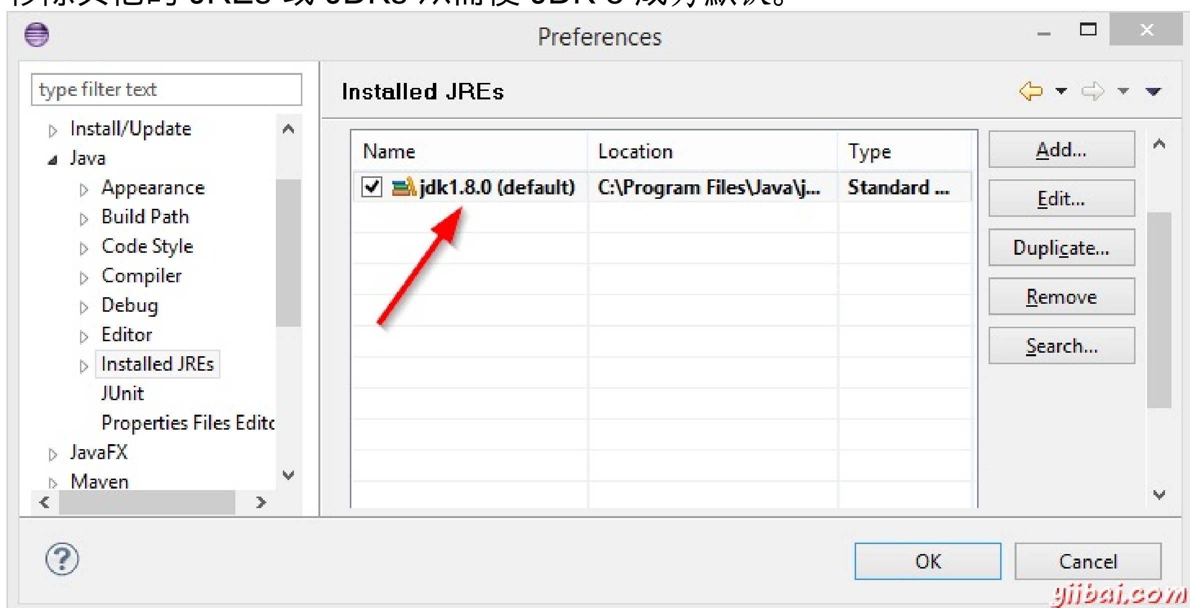
## 你需要准备

- 最新的 Java JDK 8 (包含 JavaFX 8)。
- Eclipse 4.3 或更高版本与 e(fx)clipse 插件。最简单的方法是从 e(fx)clipse 网站 下载预先配置的发行版本。作为一种备选你可以使用一个 update site 来给您的 Eclipse 安装。
- Scene Builder 2.0 或更高。

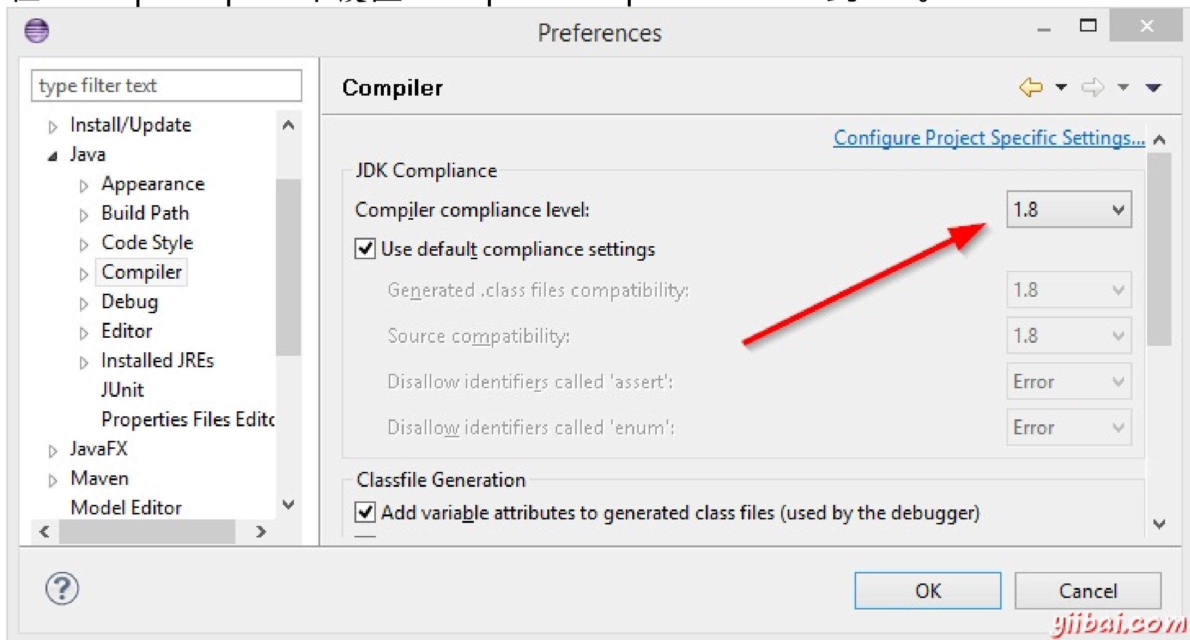
## Eclipse 配置

配置Eclipse 所使用 JDK 和 Scene Builder :

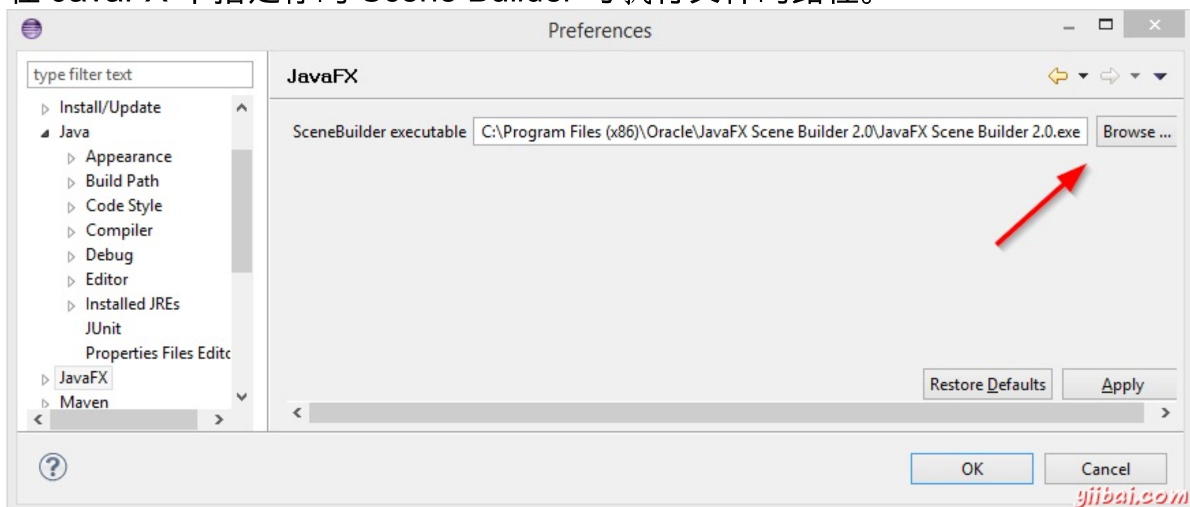
1. 打开 Eclipse 的设置并找到 Java | Installed JREs 。
2. 点击 Add..., 选择 Standard VM 并选择你安装 JDK 8 的 Directory 。
3. 移除其他的 JREs 或 JDKs 从而使 JDK 8 成为默认。



4. 在 Java | Compiler 中设置 Compiler compliance level 到 1.8。




5. 在 JavaFX 中指定你的 Scene Builder 可执行文件的路径。




## 帮助链接

你可能会想收藏下面的链接：

- Java 8 API - Java 标准类的文档。
- JavaFX 8 API - JavaFX 类的文档。
- ControlsFX API - ControlsFX project 额外 JavaFX 控件的文档。
- Oracle's JavaFX Yiibai - Oracle 的 JavaFX 官方教程。

一切就绪，让我们开始吧！

# JavaFX - 创建JavaFX项目 - JavaFX教程

## 创建一个新的 JavaFX 项目

在 Eclipse(已安装 e(fx)clipse 的)中，点击 File | New | Other... 并选择 JavaFX Project。 指定这个项目的名字(e.g. AddressApp)并点击 Finish。

如果 application 包被自动创建，那么删除它和它的内容。

### 创建包

Model-View-Controller (MVC)是一个非常重要的软件设计原则。按照MVC模式可以将我们的应用程序划分成3个部分，然后为这每一部分建立自己的包 (在源代码文件夹上右键， 选择 新建 | 包):

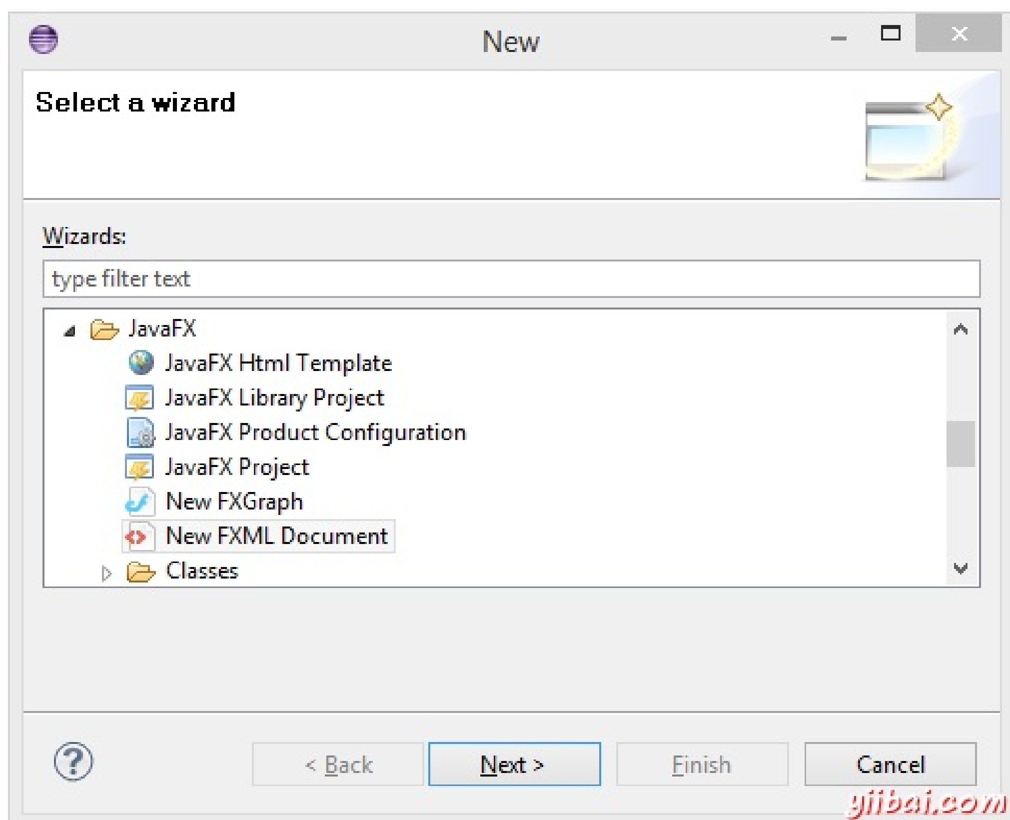
- ch.makery.address - 放置所有的控制器类(也就是应用程序的业务逻辑)
- ch.makery.address.model - 放置所有的模型类
- ch.makery.address.view - 放置所有界面和控件类

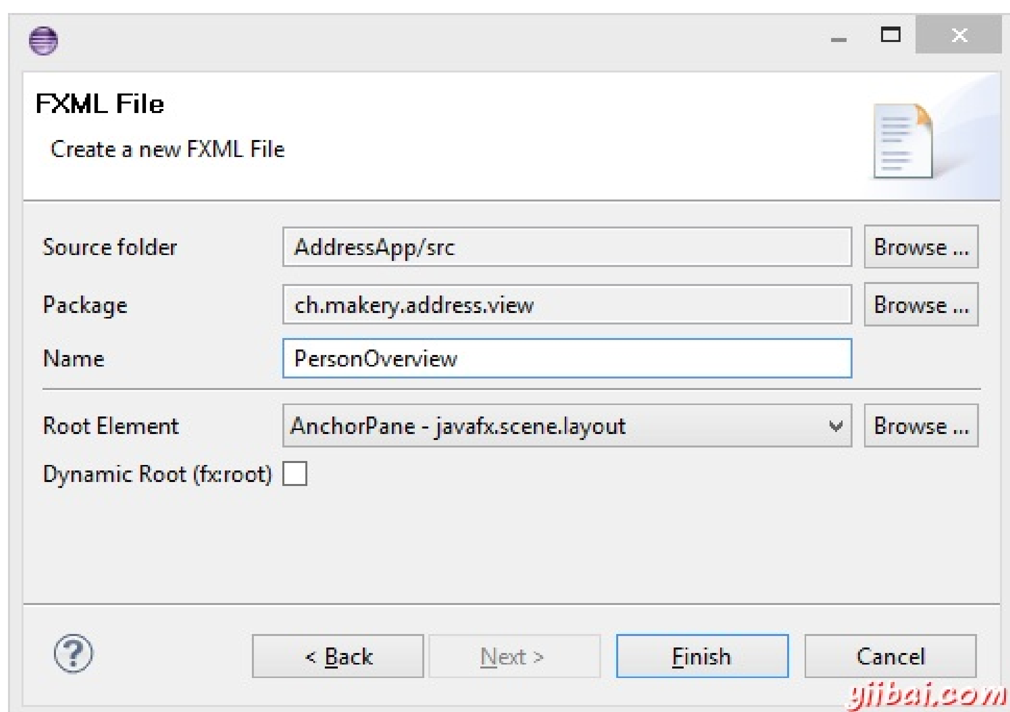
注意: view包里可能会包含一些控制器类，它可以直接被单个的view引用，我们叫它 视图-控制器。

## 创建FXML布局文件

有两种方式来创建用户界面，一终是能过XML文件来定义，另外一种就是直接通过java代码来创建. 这两种方式你都可以在网上搜到. 我们这里将使用XML的方式来创建大部分的界面。因为这种方式将会更好的将你的业务逻辑和你的界面开来，以保持代码的简洁。在接下来的内容里，我们将会介绍使用Scene Builder(所见即所得)来编辑我们的XML布局文件，它可以避免我们直接去修改XML文件。

在view包上右键创建一个新FXML Document，把它命名为PersonOverview。


New
Select a wizard
Wizards:
type filter text
JavaFX
JavaFX Html Template
JavaFX Library Project
JavaFX Product Configuration
JavaFX Project
New FXGraph
New FXML Document
Classes
< Back
Next >
Finish
Cancel
yiibai.com


FXML File
Create a new FXML File
Source folder
AddressApp/src
Browse ...
Package
ch.makery.address.view
Browse ...
Name
PersonOverview
Root Element
AnchorPane - javafx.scene.layout
Browse ...
Dynamic Root (fx:root)
< Back
Next >
Finish
Cancel
yiibai.com

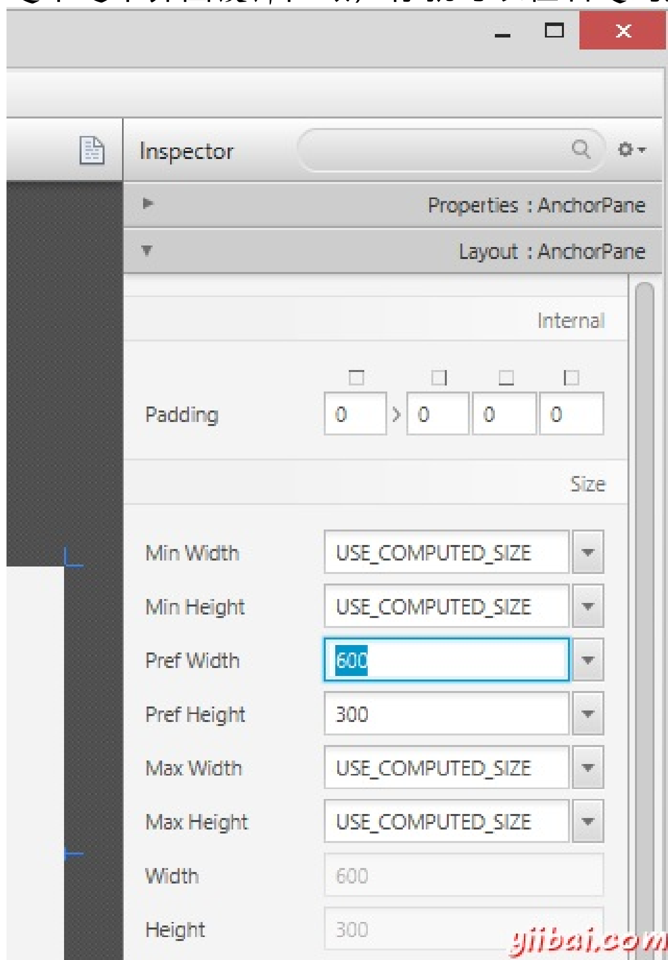
# JavaFX - Scene Builder设计界面 - JavaFX教程

## 用Scene Builder来设计你的界面

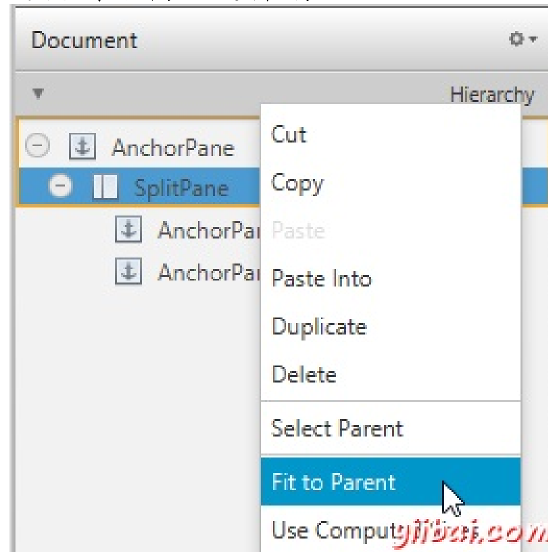
注意: 你可以下载这部分教程的源码，它里面已经包含了设计好的布局文件。

在PersonOverview.fxml 右键选择 Open with Scene Builder，那么你将会在打开的Scene Builder里面看到一个固定的界面设计区域(在整个界面的左边)。

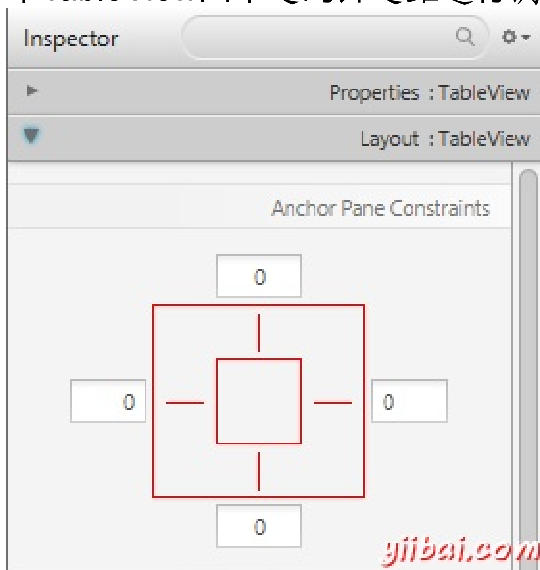
1. 选中这个界面设计区域，你就可以在右边的属性设置栏中对它的尺寸进行修改:



2. 从Scene Builder的左边控件栏中拖拽一个 Splite Pane(Horizontal Flow) 到界面设计区域，在Builder的右边视图结构中选择刚添加的Pane，在弹出的右键菜单中选择 Fit to Parent 。



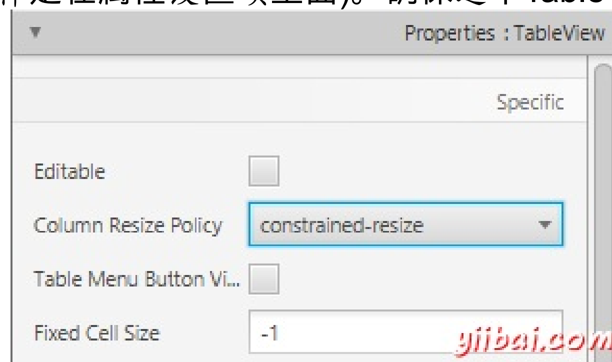

3. 同样从左边的控件栏中拖拽一个 TableView 到 SplitePane 的左边，选择这个TableView(而不是它的列)对它的布局进行设置，你可以在 AnchorPane 中对这个TableView四个边的外边距进行调节。(more information on Layouts).

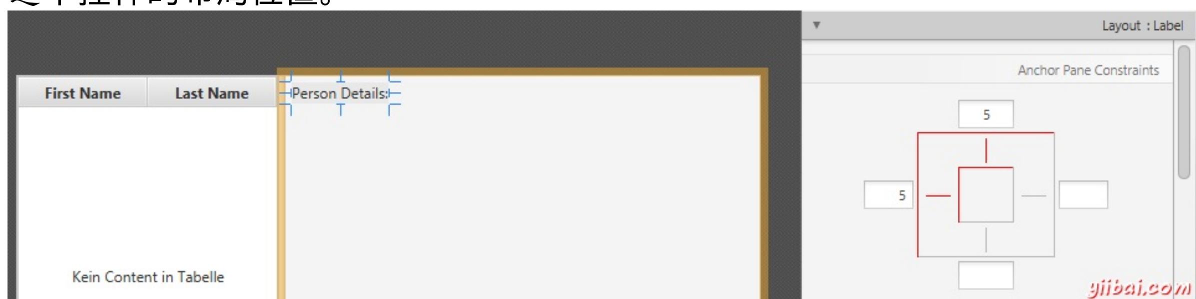



4. 点击菜单中的 Preview | Show Preview in Window 可以预览你设计好的界面，试着缩放预览的界面，你会发现TableView会随着窗口的缩放而变化。

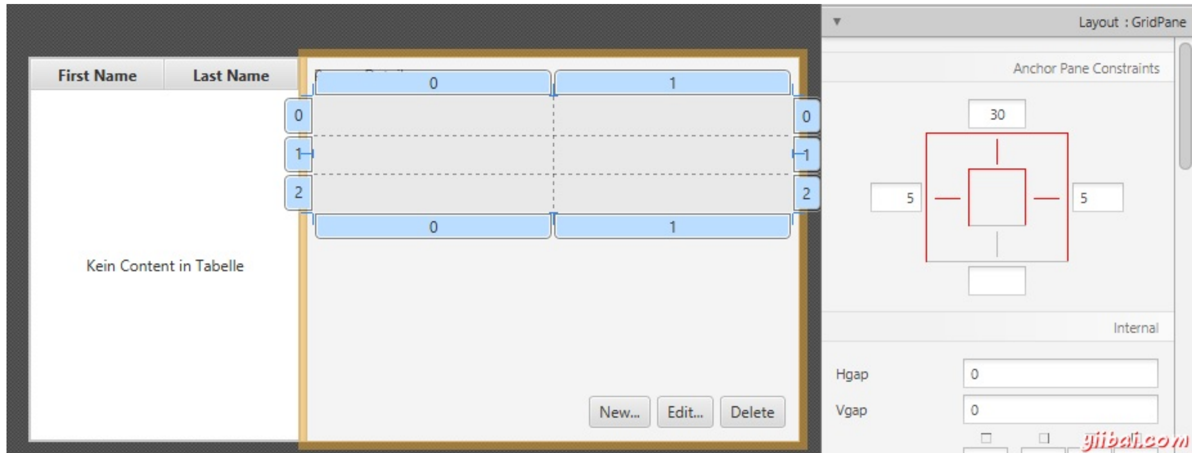
5. 修改TableView中的列名字，"First Name" and "Last Name"，在右边面板中的属性设置项



6. 选择这个 TableView， 在右边面板中将它的 Column Resize Policy 修改成 constrained-resize (同样是在属性设置项里面)。确保这个TableView的列能够铺满所有的可用空间。


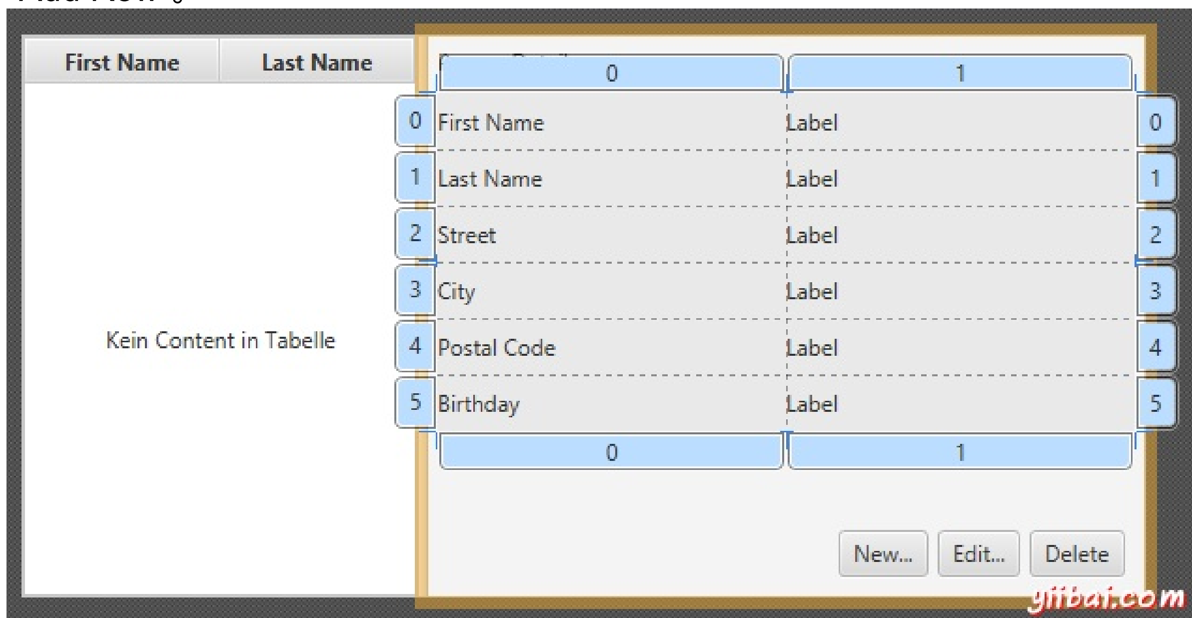

7. 添加一个 Label 到 SplitePane的右边部分，并设置它的显示文字为 "Person Details" (提示: 你可以通过搜索来找到 Label 这个控件)。 使用anchors来调节这个控件的布局位置。


8. 再添加一个 GridPane SplitePane的右边部分, 使用anchors来调节这个控件的布局位置。




9. 按照下面的图添加多个 Lables到表格中去。

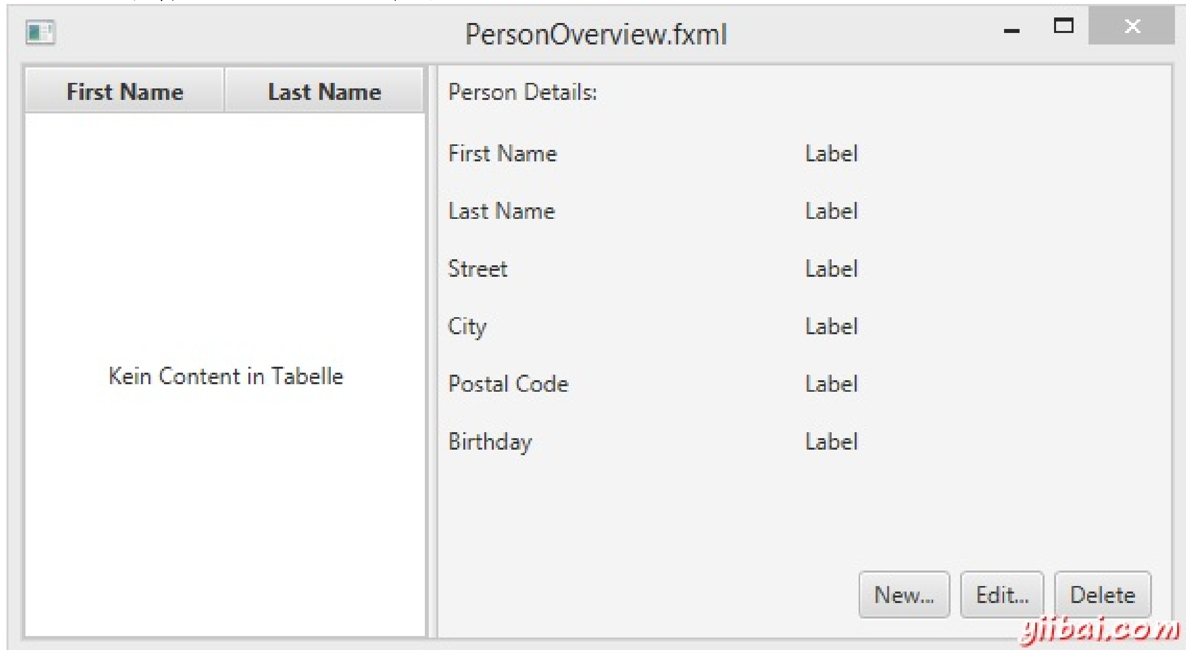
10. 注意: 添加一个控件到已经存在的行里面去，你可在这行的行号上右键选择"Add Row"。




11. 添加3个按钮到这个 GridPane 的下面。 小提示: 选择这3个按钮，右键 Wrap In | HBox，那么它们会被放置到一个HBox里面。 你可能需要对这个HBox指定一个 spacing,同时也需要设置它们的右边和下边的anchors。



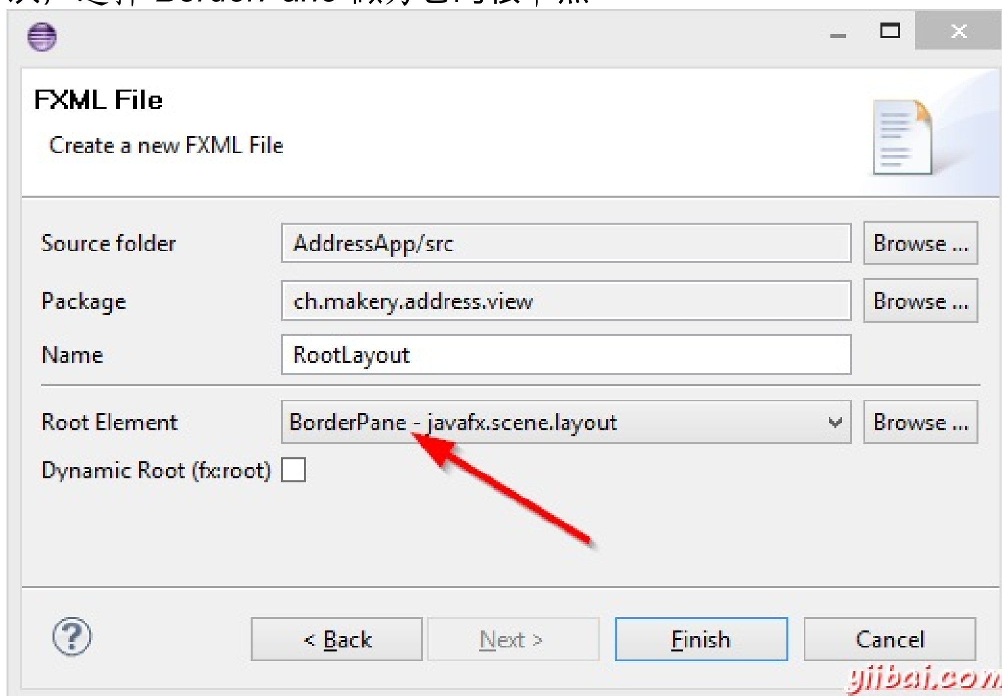
12. 那么基本已经完成了界面的设计，你可以通过 Preview 来预览一下你设计的界面，同时缩放一下窗口来检验一下各个控件的位置是否正确。

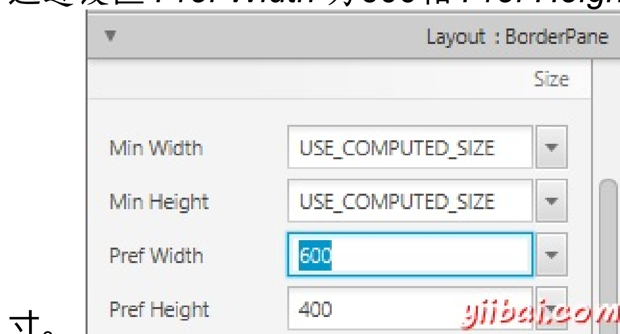


# JavaFX - 创建主应用程序 - JavaFX教程

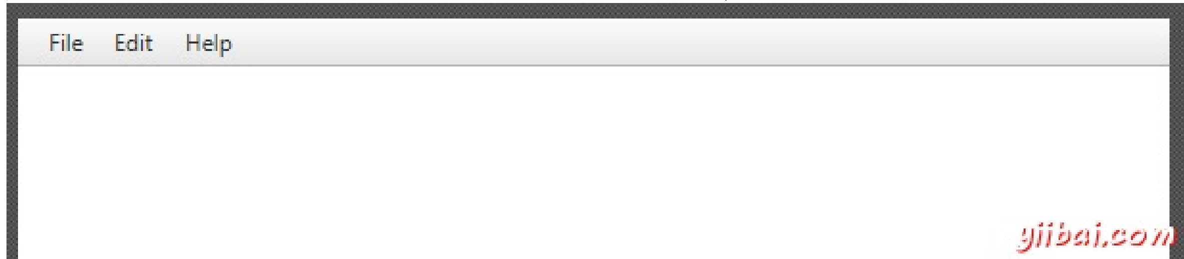
## 创建主应用程序

我们需要建立一个新的布局文件 `PersonOverview.fxml` 来做为主布局文件，它将包含一个菜单栏和你即将要显示的布局。

1. 在view包里面创建一个新的 *FXML Document* 叫做 `RootLayout.fxml` ，这一次，选择 *BorderPane* 做为它的根节点


2. 在Scene Builder中打开 `RootLayout.fxml` 。
3. 通过设置 *Pref Width* 为600和 *Pref Height* 为400来改变这个 *BorderPane*的尺寸。


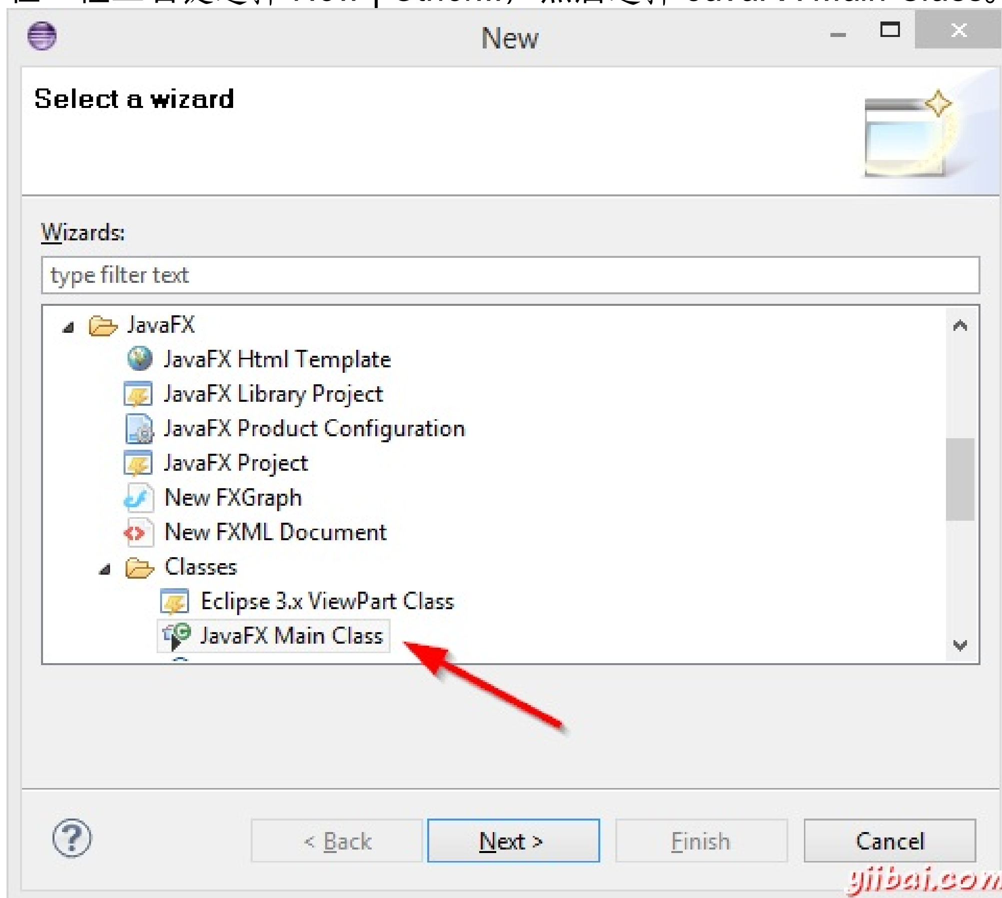
4. 在最顶上添加一个 *MenuBar*，先不去给这个菜单添加任何的功能。



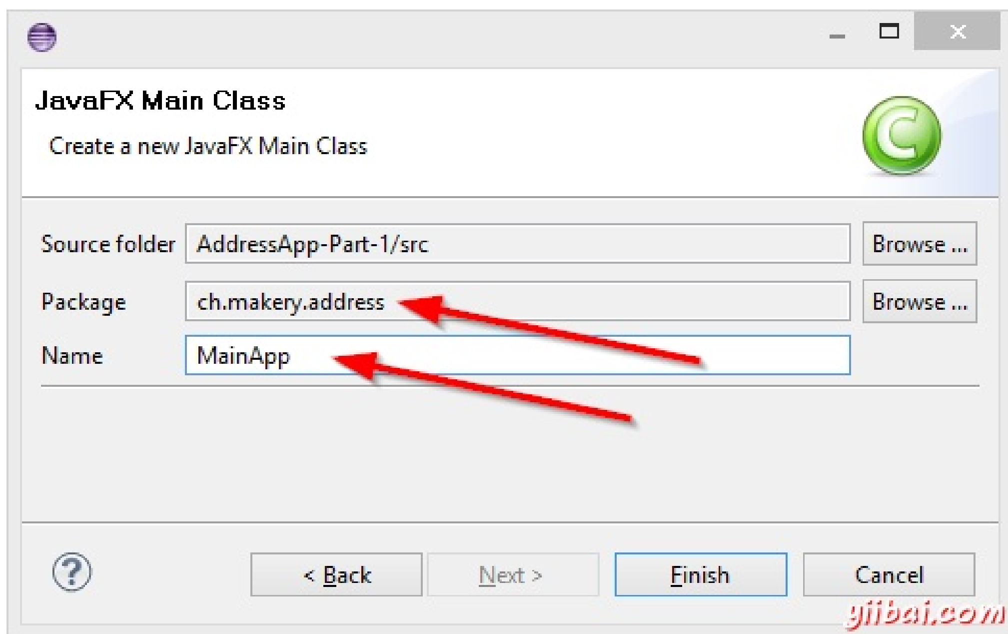

## The JavaFX Main Class

现在，我们需要创建一个 main java class 用来加载 `RootLayout.fxml` ，同时添加 `PersonOverview.fxml` 到*RootLayout.fxml*中去，这个main class将做为我们这个应用程序的入口。

1. 在工程上右键选择 *New | Other...*，然后选择 *JavaFX Main Class*。




2. 将这个class命名为 `MainApp` ，将它放置到controller包中，也就是上面建的 `ch.makery.address` (注意: 这个包下有两个子包，分别是 `view` 和 `model` )。




你可能注意到了IDE生成的 `MainApp.java` 继承自 `Application` 同时包含了两个方法， 这是一个JavaFX应用程序的最基本的代码结构，这里最重要的方法是 `start(Stage primaryStage)` ，它将会在应用程序运行时通过内部的 `main` 方法自动调用。

正如你所看到的，这个 `start(...)` 方法会接收一个 `Stage` 类型的参数，下面的图向你展示了一个JavaFX应用程序的基本结构。

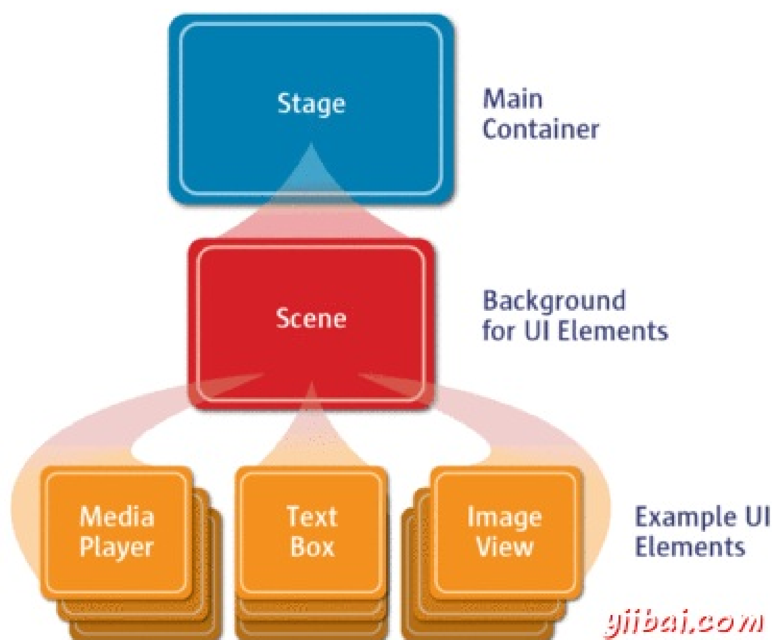



*Image Source: http://www.oracle.com*

一切看起来象是剧场里表演: 这里的 `Stage` 是一个主容器，它就是我们通常所认为的窗口(有边，高和宽，还有关闭按钮)。在这个 `Stage` 里面，你可以放置一个 `Scene` ，当然你可以切换别的 `Scene` ，而在这个 `Scene` 里面，我们就可以放置各种各样的控件。

更详细的信息，你可以参考 Working with the JavaFX Scene Graph.

打开 `MainApp.java` ，将已有的代码替换成下面的代码:

```
package ch.makery.address;

import java.io.IOException;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MainApp extends Application {

    private Stage primaryStage;
    private BorderPane rootLayout;

    @Override
    public void start(Stage primaryStage) {
        this.primaryStage = primaryStage;
        this.primaryStage.setTitle("AddressApp");

        initRootLayout();

        showPersonOverview();
    }

    /**
     * Initializes the root layout.
     */
    public void initRootLayout() {
        try {
            // Load root layout from fxml file.
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(MainApp.class.getResource("view/Root
            rootLayout = (BorderPane) loader.load();

            // Show the scene containing the root layout.
            Scene scene = new Scene(rootLayout);
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Shows the person overview inside the root layout.
     */
    public void showPersonOverview() {
        try {
```

```
            // Load person overview.
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(MainApp.class.getResource("view/Pers
            AnchorPane personOverview = (AnchorPane) loader.load();

            // Set person overview into the center of root layout.
            rootLayout.setCenter(personOverview);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Returns the main stage.
     * @return
     */
    public Stage getPrimaryStage() {
        return primaryStage;
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

代码中的注释会给你一些小提示，注明代码的含义。

如果你现在就运行这个程序，那么你将会看到和这篇文章开头所展示的图片那样的界面。

## 你有可能遇见的问题

如果你的应用程序找不到你所指定的 `fxml` 布局文件，那么系统会提示以下的错误：

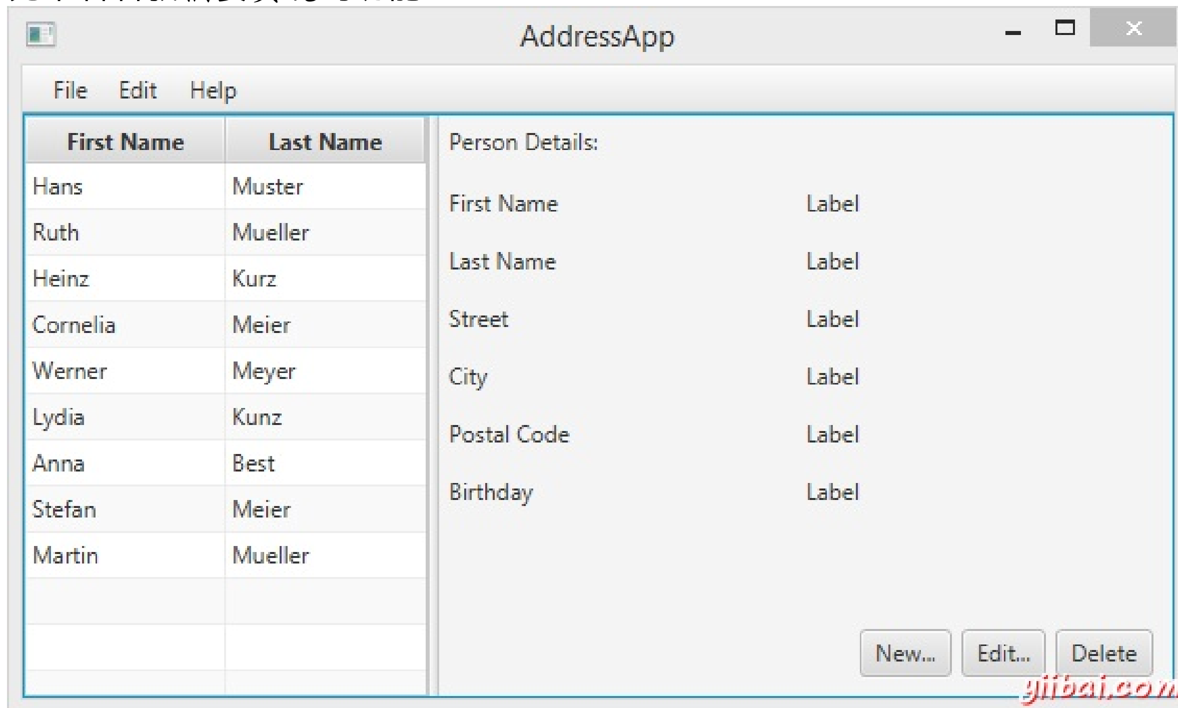
```
java.lang.IllegalStateException: Location is not set.
```

你可以检查一下你的 `fxml` 文件名是否拼写错误

如果还是不能工作，请下载这篇教程所对应的源代码，然后将源代码中的fxml文件替换掉你的

# JavaFX - Model和TableView - JavaFX教程

先来看看我们要实现的功能：




## 第二部分的主题

- 创建一个 模型 类。
- 在 ObservableList 使用模型类。
- 使用 Controllers 在 TableView 上显示数据。

## 创建 模型 类。

我们需要一个模型类来保存联系人信息到我们的通讯录中。在模型包中 ( `ch.makery.address.model` ) 添加一个叫 `Person` 的类。 `Person` 类将会有一些变量，名字，地址和生日。将以下代码添加到类。在代码后，我将解释一些 JavaFX 的细节。

### Person.java

```
package ch.makery.address.model;

import java.time.LocalDate;

import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleIntegerProperty;
```

```
import javafx.beans.property.SimpleObjectProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

/**
 * Model class for a Person.
 *
 * @author Marco Jakob
 */
public class Person {

    private final StringProperty firstName;
    private final StringProperty lastName;
    private final StringProperty street;
    private final IntegerProperty postalCode;
    private final StringProperty city;
    private final ObjectProperty<LocalDate> birthday;

    /**
     * Default constructor.
     */
    public Person() {
        this(null, null);
    }

    /**
     * Constructor with some initial data.
     *
     * @param firstName
     * @param lastName
     */
    public Person(String firstName, String lastName) {
        this.firstName = new SimpleStringProperty(firstName);
        this.lastName = new SimpleStringProperty(lastName);

        // Some initial dummy data, just for convenient testing.
        this.street = new SimpleStringProperty("some street");
        this.postalCode = new SimpleIntegerProperty(1234);
        this.city = new SimpleStringProperty("some city");
        this.birthday = new SimpleObjectProperty<LocalDate>(LocalDa
    }

    public String getFirstName() {
        return firstName.get();
    }

    public void setFirstName(String firstName) {
        this.firstName.set(firstName);
    }

    public StringProperty firstNameProperty() {
        return firstName;
    }
```

```
    public String getLastName() {
        return lastName.get();
    }

    public void setLastName(String lastName) {
        this.lastName.set(lastName);
    }

    public StringProperty lastNameProperty() {
        return lastName;
    }

    public String getStreet() {
        return street.get();
    }

    public void setStreet(String street) {
        this.street.set(street);
    }

    public StringProperty streetProperty() {
        return street;
    }

    public int getPostalCode() {
        return postalCode.get();
    }

    public void setPostalCode(int postalCode) {
        this.postalCode.set(postalCode);
    }

    public IntegerProperty postalCodeProperty() {
        return postalCode;
    }

    public String getCity() {
        return city.get();
    }

    public void setCity(String city) {
        this.city.set(city);
    }

    public StringProperty cityProperty() {
        return city;
    }

    public LocalDate getBirthday() {
        return birthday.get();
    }
```

```
    public void setBirthday(LocalDate birthday) {
        this.birthday.set(birthday);
    }

    public ObjectProperty<LocalDate> birthdayProperty() {
        return birthday;
    }
}
```

## 解释

- 在JavaFX中,对一个模型类的所有属性使用 `Properties` 是很常见的. 一个 `Property` 允许我们, 打个比方, 当 `lastName` 或其他属性被改变时自动收到通知, 这有助于我们保持视图与数据的同步, 阅读 Using JavaFX Properties and Binding 学习更多关于 `Properties` 的内容。
- `LocalDate` , 我们使用了 `birthday` 类型, 这是一个新的部分在 Date and Time API for JDK 8.

# 人员列表

我们的应用主要管理的数据是一群人的信息.让我们在 `MainApp` 类里面创建一个 `Person` 对象的列表。稍后其他所有的控制器类将存取 `MainApp` 的核心列表。

## ObservableList

我们处理JavaFX的view classes需要在人员列表发生任何改变时都被通知. 这是很重要的,不然视图就会和数据不同步.为了达到这个目的,JavaFX引入了一些新的集合类.

在这些集合中, 我们需要的是 `ObservableList` . 将以下代码增加到 `MainApp` 类的开头去创建一个新的 `ObservableList` . 我们也会增加一个构造器去创建一些样本数据和一个公共的getter方法:

**MainApp.java**

```
    // ... AFTER THE OTHER VARIABLES ...

    /**
     * The data as an observable list of Persons.
     */
    private ObservableList<Person> personData = FXCollections.obser

    /**
     * Constructor
     */
    public MainApp() {
        // Add some sample data
        personData.add(new Person("Hans", "Muster"));
        personData.add(new Person("Ruth", "Mueller"));
        personData.add(new Person("Heinz", "Kurz"));
        personData.add(new Person("Cornelia", "Meier"));
        personData.add(new Person("Werner", "Meyer"));
        personData.add(new Person("Lydia", "Kunz"));
        personData.add(new Person("Anna", "Best"));
        personData.add(new Person("Stefan", "Meier"));
        personData.add(new Person("Martin", "Mueller"));
    }

    /**
     * Returns the data as an observable list of Persons.
     * @return
     */
    public ObservableList<Person> getPersonData() {
        return personData;
    }

    // ... THE REST OF THE CLASS ...
```

## The PersonOverviewController

现在我们终于要将数据加入到表格中了,我们需要一个控制器为了 `PersonOverview.fxml` ,.

1. 在view包下创建一个名为 `PersonOverviewController.java` 的普通java类(我们需要将这个类放在和 `PersonOverview.fxml` 相同的包下, 不然SceneBuilder会找不到它 - 至少在当前的版本).
2. 我们需要增加一些实例变量来访问表格和在视图中的标签.这些属性和一些方法有一个特殊的 `@FXML` 注解. 这对于fxml文件访问私有属性和私有方法来说是必需的. 当将一切都在fxml文件中设置好之后, 应用程序会在fxml文件被载入时自动地填充这些变量. 让我们添加以下的代码:

Note: 记住要使用 javafx imports, 而不是awt和swing!

### PersonOverviewController.java

```
package ch.makery.address.view;

import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import ch.makery.address.MainApp;
import ch.makery.address.model.Person;

public class PersonOverviewController {
    @FXML
    private TableView<Person> personTable;
    @FXML
    private TableColumn<Person, String> firstNameColumn;
    @FXML
    private TableColumn<Person, String> lastNameColumn;

    @FXML
    private Label firstNameLabel;
    @FXML
    private Label lastNameLabel;
    @FXML
    private Label streetLabel;
    @FXML
    private Label postalCodeLabel;
    @FXML
    private Label cityLabel;
    @FXML
    private Label birthdayLabel;

    // Reference to the main application.
    private MainApp mainApp;

    /**
     * The constructor.
     * The constructor is called before the initialize() method.
     */
    public PersonOverviewController() {
    }

    /**
     * Initializes the controller class. This method is automatical
     * after the fxml file has been loaded.
     */
    @FXML
    private void initialize() {
        // Initialize the person table with the two columns.
        firstNameColumn.setCellValueFactory(cellData -> cellData.ge
        lastNameColumn.setCellValueFactory(cellData -> cellData.get
    }
```

```
    /**
     * Is called by the main application to give a reference back t
     *
     * @param mainApp
     */
    public void setMainApp(MainApp mainApp) {
        this.mainApp = mainApp;

        // Add observable list data to the table
        personTable.setItems(mainApp.getPersonData());
    }
}
```

可能需要解释一下这段代码:

- 所有fxml文件需要访问的属性和方法必须加上 `@FXML` 注解.实际上,只有在私有的情况下才需要, 但是让它们保持私有并且用注解标记的方式更好!
- `initialize()` 方法在fxml文件完成载入时被自动调用. 那时, 所有的FXML属性都应已被初始化.
- 我们在表格列上使用 `setCellValueFactory(...)` 来确定为特定列使用 `Person` 对象的某个属性. 箭头 `->` 表示我们在使用Java 8的 *Lambdas* 特性. (另一个选择是使用 PropertyValueFactory, 但它不是类型安全的).

## 连接 MainApp 和 PersonOverviewController

`setMainApp(...)` 必须被 `MainApp` 类调用. 这让我们可以访问 `MainApp` 对象并得到 `Persons` 的列表和其他东西. 用以下代码替换 `showPersonOverview()` 方法. 它包含了新增的两行:

**MainApp.java - new showPersonOverview() method**

```
/**
 * Shows the person overview inside the root layout.
 */
public void showPersonOverview() {
    try {
        // Load person overview.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/PersonOv
        AnchorPane personOverview = (AnchorPane) loader.load();

        // Set person overview into the center of root layout.
        rootLayout.setCenter(personOverview);

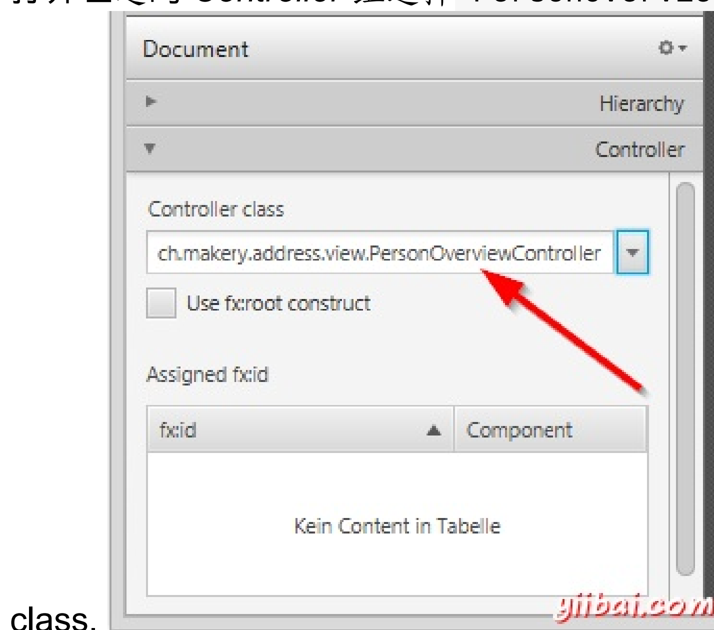
        // Give the controller access to the main app.
        PersonOverviewController controller = loader.getController(
        controller.setMainApp(this);

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

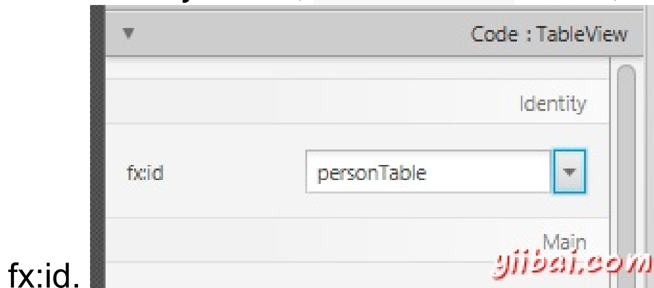
## 将View与Controller挂钩

我们快要完成了! 但是有件小事被遗漏了: 至今没有告诉 `PersonOverview.fxml` 使用的是哪个控制器以及元素与控制器中的属性的对应关系.

1. 使用*SceneBuilder* 打开 `PersonOverview.fxml` .
2. 打开左边的 *Controller* 组选择 `PersonOverviewController` 作为 controller class.



3. 在 *Hierarchy* 组选择 `TableView` 并选择 *Code* 组将 `personTable` 作为 fx:id.




4. 对列做相同的事并且将 `firstNameColumn` and `lastNameColumn` 分别作为 fx:id .
5. 对在第二列的 each label , 选择对应的 fx:id.

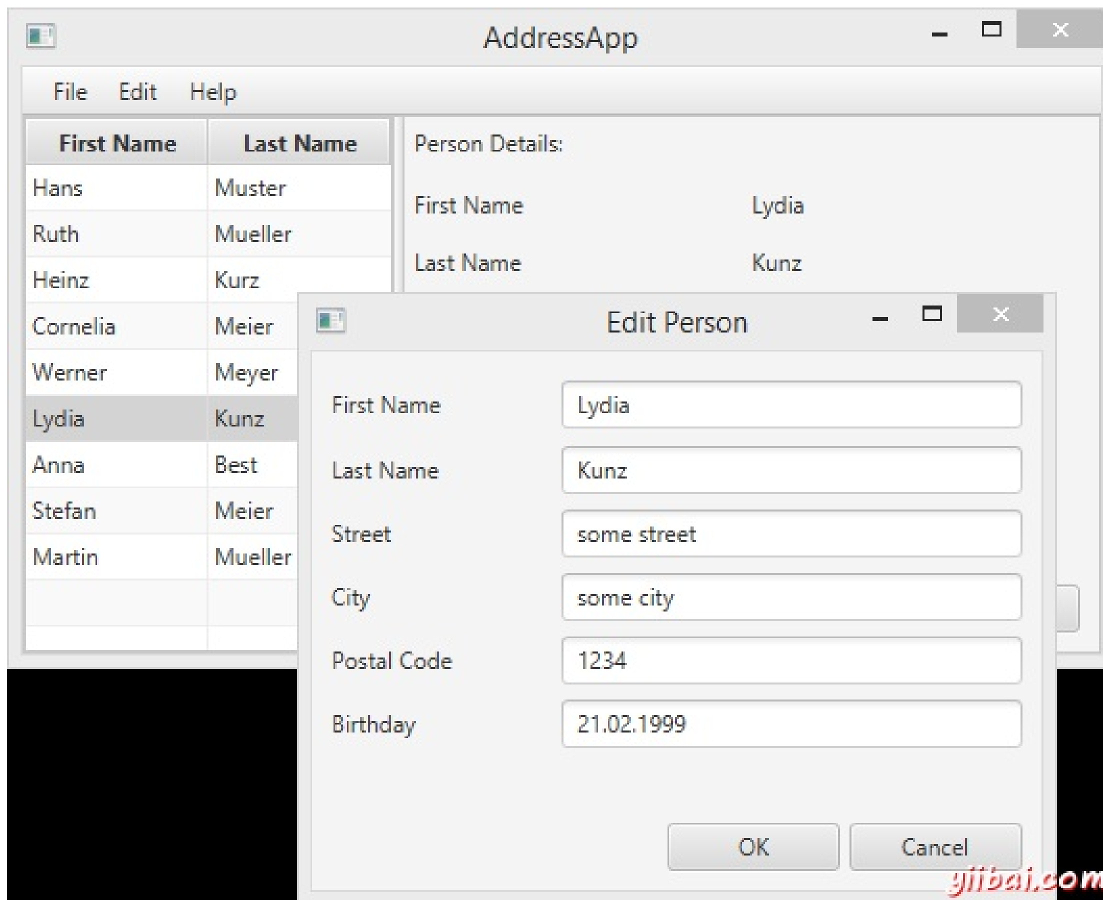



6. 重要事项: 回到eclipse并且 refresh the entire AddressApp project (F5). 这是必要的因为有时候eclipse并不知道在Scene Builder中作出的改变.

# 启动应用程序

当你现在启动了你的应用,你应该看到了类似这篇博客开头的截图的程序界面.

# JavaFX - 用户交互 - JavaFX教程




## 第3部分的主题：

1. 在表中反应选择的改变(TableView中)。
2. 增加增加，编辑和删除按钮的功能。
3. 创建自定义弹出对话框编辑人员。
4. 验证用户输入。

## 响应表的选择

显然，我们还没有使用应用程序的右边。想法是当用户选择表中的人员时，在右边显示人员的详情。

首先，让我们在 `PersonOverviewController` 添加一个新的方法，帮助我们使用单个人员的数据填写标签。

创建方法 `showPersonDetails(Person person)` 。遍历所有标签，并且使用 `setText(…)` 方法设置标签的文本为个人的详情。如果null作为参数传递，所有的标签应该被清空。

**PersonOverviewController.java**

```
/**
 * Fills all text fields to show details about the person.
 * If the specified person is null, all text fields are cleared.
 *
 * @param person the person or null
 */
private void showPersonDetails(Person person) {
    if (person != null) {
        // Fill the labels with info from the person object.
        firstNameLabel.setText(person.getFirstName());
        lastNameLabel.setText(person.getLastName());
        streetLabel.setText(person.getStreet());
        postalCodeLabel.setText(Integer.toString(person.getPostalCo
        cityLabel.setText(person.getCity());

        // TODO: We need a way to convert the birthday into a Strin
        // birthdayLabel.setText(...);
    } else {
        // Person is null, remove all the text.
        firstNameLabel.setText("");
        lastNameLabel.setText("");
        streetLabel.setText("");
        postalCodeLabel.setText("");
        cityLabel.setText("");
        birthdayLabel.setText("");
    }
}
```

## 转换生日日期为字符串

你注意到我们没有设置 `birthday` 到标签中，因为它是 `LocalDate` 类型，不是 `String` 。我们首先需要格式化日期。

在几个地方上我们使用 `LocalDate` 和 `String` 之间的转换。好的实践是创建一个带有 `static` 方法的帮助类。我们称它为 `DateUtil` ，并且把它放到单独的包中，称为 `ch.makery.address.util` 。

**DateUtil.java**

```
package ch.makery.address.util;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

/**
```

```
 * Helper functions for handling dates.
 *
 * @author Marco Jakob
 */
public class DateUtil {

    /** The date pattern that is used for conversion. Change as you
    private static final String DATE_PATTERN = "dd.MM.yyyy";

    /** The date formatter. */
    private static final DateTimeFormatter DATE_FORMATTER =
            DateTimeFormatter.ofPattern(DATE_PATTERN);

    /**
     * Returns the given date as a well formatted String. The above
     * {@link DateUtil#DATE_PATTERN} is used.
     *
     * @param date the date to be returned as a string
     * @return formatted string
     */
    public static String format(LocalDate date) {
        if (date == null) {
            return null;
        }
        return DATE_FORMATTER.format(date);
    }

    /**
     * Converts a String in the format of the defined {@link DateUt
     * to a {@link LocalDate} object.
     *
     * Returns null if the String could not be converted.
     *
     * @param dateString the date as String
     * @return the date object or null if it could not be convertec
     */
    public static LocalDate parse(String dateString) {
        try {
            return DATE_FORMATTER.parse(dateString, LocalDate::fron
        } catch (DateTimeParseException e) {
            return null;
        }
    }

    /**
     * Checks the String whether it is a valid date.
     *
     * @param dateString
     * @return true if the String is a valid date
     */
    public static boolean validDate(String dateString) {
        // Try to parse the String.
        return DateUtil.parse(dateString) != null;
```

```
    }
}
```

提示：你能通过改变 `DATE_PATTERN` 修改日期的格式。所有可能的格式参考 DateTimeFormatter.

## 使用DateUtil

现在，我们需要在 `PersonOverviewController` 的 `showPersonDetails` 方法中使用我们新建的 `DateUtil` 。使用下面这样替代我们添加的 `TODO` 。

```
birthdayLabel.setText(DateUtil.format(person.getBirthday()));
```

## 监听表选择的改变

为了当用户在人员表中选择一个人时获得通知，我们需要监听改变。

在JavaFX中有一个接口称为 `ChangeListener` ，带有一个方法 `changed()` 。该方法有三个参数： `observable` ， `oldValue` 和 `newValue` 。

我们使用*Java 8 lambda*表达式创建这样一个 `ChangeListener` 。让我们添加一些行到 `PersonOverviewController` 的 `initialize()` 方法中。现在看起来是这样的。

### PersonOverviewController.java

```
@FXML
private void initialize() {
    // Initialize the person table with the two columns.
    firstNameColumn.setCellValueFactory(
            cellData -> cellData.getValue().firstNameProperty());
    lastNameColumn.setCellValueFactory(
            cellData -> cellData.getValue().lastNameProperty());

    // Clear person details.
    showPersonDetails(null);

    // Listen for selection changes and show the person details whe
    personTable.getSelectionModel().selectedItemProperty().addListe
            (observable, oldValue, newValue) -> showPersonDetails(n
}
```

使用 `showPersonDetails(null)` ，我们重设个人详情。

使用 `personTable.getSelectionModel...` ，我们获得人员表的 `selectedItemProperty` ，并且添加监听。不管什么时候用户选择表中的人员，都会执行我们的 `lambda` 表达式。我们获取新选择的人员，并且把它传递给 `showPersonDetails(...)` 方法。

现在试着运行你的应用程序，验证当你选择表中的人员时，关于该人员的详情是否正确的显示。

如果有些事情不能工作，你可以对比下PersonOverviewController.java中的 `PersonOverviewController` 类

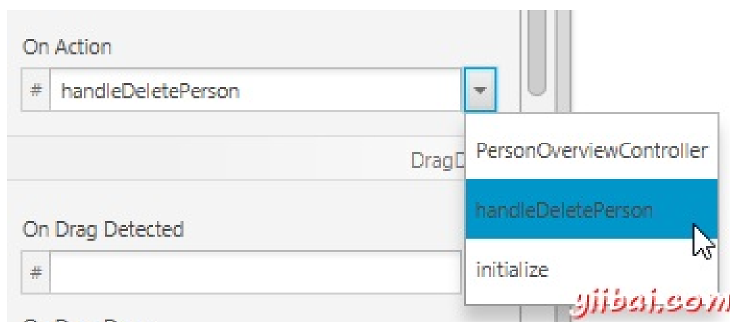
## 删除按钮

我们的用户接口已经包含一个删除按钮，但是没有任何功能。我们能在*SceneBuilder*中的按钮上选择动作。在我们控制器中的任何使用 `@FXML` (或者它是公用的)注释的方法都可以被*Scene Builder*访问。因此，让我们在 `PersonOverviewController` 类的最后添加一个删除方法。

**PersonOverviewController.java**

```
/**
 * Called when the user clicks on the delete button.
 */
@FXML
private void handleDeletePerson() {
    int selectedIndex = personTable.getSelectionModel().getSelectec
    personTable.getItems().remove(selectedIndex);
}
```

现在，使用*SceneBuilder*打开 `PersonOverview.fxml` 文件，选择*Delete*按钮，打开*Code*组，在On Actin的下拉菜单中选择 `handleDeletePerson` 。




## 错误处理

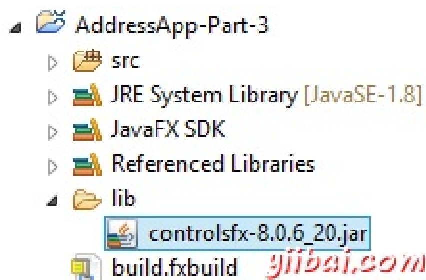
如果你现在运行应用程序，你应该能够从表中删除选择的人员。但是，当你没有在表中选择人员时点击删除按钮时会发生什么呢。

这里有一个 `ArrayIndexOutOfBoundsException` ，因为它不能删除掉索引为-1人员项目。索引-1由 `getSelectedIndex()` 返回，它意味着你没有选择项目。

当然，忽略这种错误不是非常好。我们应该让用户知道在删除时必须选择一个人员。(更好的是我们应该禁用删除按钮，以便用户没有机会做错误的事情)。

我们添加一个弹出对话框通知用户，你将需要*添加一个库Dialogs：

1. 下载controlsfx-8.0.6_20.jar (你也能从ControlsFX Website中获取)。 重要：ControlsFX必须是8.0.6_20以上版本才能在 `JDK8U20` 以上版本工作。
2. 在项目中创建一个lib子目录，添加controlsf jar文件到该目录下。
3. 添加库到你的项目classpath中。在Eclipse中右击*jar*文件|选择*Build Path*| *Add to Build Path*。现在Eclipse知道这个库了。




对 `handleDeletePerson()` 方法做一些修改后，不管什么时候用户没有选择表中的人员时按下删除按钮，我们能显示一个简单的对话框。

**PersonOverviewController.java**

```
/**
 * Called when the user clicks on the delete button.
 */
@FXML
private void handleDeletePerson() {
    int selectedIndex = personTable.getSelectionModel().getSelectec
    if (selectedIndex >= 0) {
        personTable.getItems().remove(selectedIndex);
    } else {
        // Nothing selected.
        Dialogs.create()
            .title("No Selection")
            .masthead("No Person Selected")
            .message("Please select a person in the table.")
            .showWarning();
    }
}
```

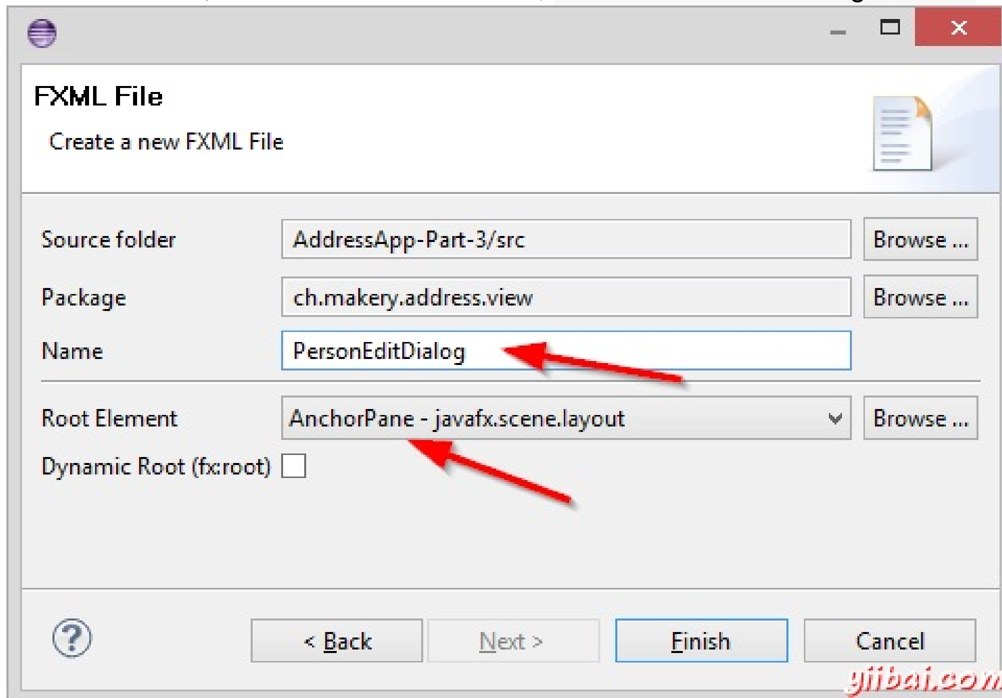
更多如何使用Dialog的示例，请阅读JavaFX 8 Dialogs.

## 新建和编辑对话框

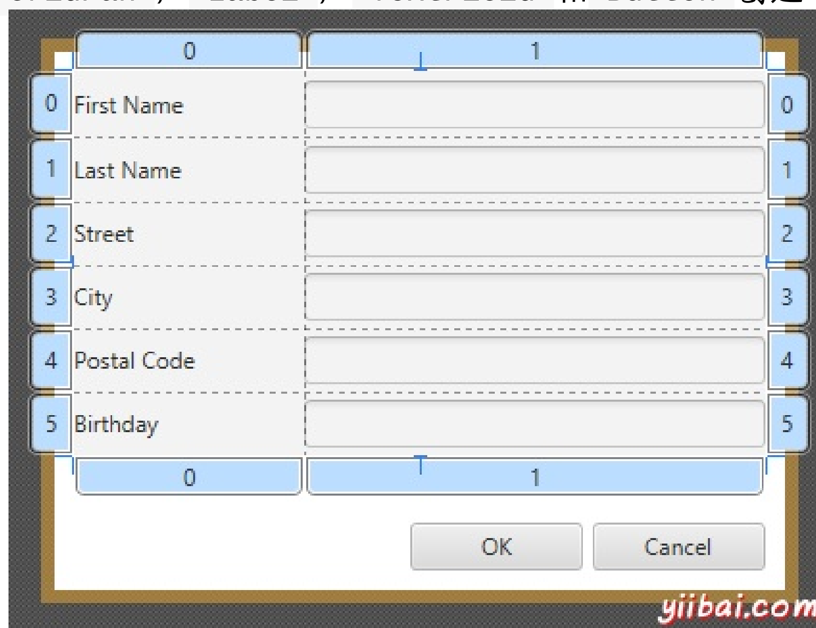
新建和编辑的动作有点工作：我们需要一个自定义带表单的对话框(例如：新的Stage)，询问用户关于人员的详情。

## 设计对话框

1. 在*view*包中创建新的fxml文件，称为 `PersonEditDialog.fxml`




2. 使用 `GridPan` ， `Label` ， `TextField 和 Button` 创建一个对话框，如下所示:




如果你不能完成工作，你能下载这个*PersonEditDialog.fxml*.

## 创建控制器

为对话框创建控制器 `PersonEditDialogController.java` :

**PersonEditDialogController.java**

```
package ch.makery.address.view;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import org.controlsfx.dialog.Dialogs;

import ch.makery.address.model.Person;
import ch.makery.address.util.DateUtil;

/**
 * Dialog to edit details of a person.
 *
 * @author Marco Jakob
 */
public class PersonEditDialogController {

    @FXML
    private TextField firstNameField;
    @FXML
    private TextField lastNameField;
    @FXML
    private TextField streetField;
    @FXML
    private TextField postalCodeField;
    @FXML
    private TextField cityField;
    @FXML
    private TextField birthdayField;

    private Stage dialogStage;
    private Person person;
    private boolean okClicked = false;

    /**
     * Initializes the controller class. This method is automatical
     * after the fxml file has been loaded.
     */
    @FXML
    private void initialize() {
    }

    /**
     * Sets the stage of this dialog.
     *
     * @param dialogStage
     */
    public void setDialogStage(Stage dialogStage) {
        this.dialogStage = dialogStage;
```

```
    }

    /**
     * Sets the person to be edited in the dialog.
     *
     * @param person
     */
    public void setPerson(Person person) {
        this.person = person;

        firstNameField.setText(person.getFirstName());
        lastNameField.setText(person.getLastName());
        streetField.setText(person.getStreet());
        postalCodeField.setText(Integer.toString(person.getPostalCo
        cityField.setText(person.getCity());
        birthdayField.setText(DateUtil.format(person.getBirthday()
        birthdayField.setPromptText("dd.mm.yyyy");
    }

    /**
     * Returns true if the user clicked OK, false otherwise.
     *
     * @return
     */
    public boolean isOkClicked() {
        return okClicked;
    }

    /**
     * Called when the user clicks ok.
     */
    @FXML
    private void handleOk() {
        if (isInputValid()) {
            person.setFirstName(firstNameField.getText());
            person.setLastName(lastNameField.getText());
            person.setStreet(streetField.getText());
            person.setPostalCode(Integer.parseInt(postalCodeField.g
            person.setCity(cityField.getText());
            person.setBirthday(DateUtil.parse(birthdayField.getText

            okClicked = true;
            dialogStage.close();
        }
    }

    /**
     * Called when the user clicks cancel.
     */
    @FXML
    private void handleCancel() {
        dialogStage.close();
    }
```

```
    /**
     * Validates the user input in the text fields.
     *
     * @return true if the input is valid
     */
    private boolean isInputValid() {
        String errorMessage = "";

        if (firstNameField.getText() == null || firstNameField.getT
            errorMessage += "No valid first name!\n";
        }
        if (lastNameField.getText() == null || lastNameField.getTex
            errorMessage += "No valid last name!\n";
        }
        if (streetField.getText() == null || streetField.getText().
            errorMessage += "No valid street!\n";
        }

        if (postalCodeField.getText() == null || postalCodeField.ge
            errorMessage += "No valid postal code!\n";
        } else {
            // try to parse the postal code into an int.
            try {
                Integer.parseInt(postalCodeField.getText());
            } catch (NumberFormatException e) {
                errorMessage += "No valid postal code (must be an i
            }
        }

        if (cityField.getText() == null || cityField.getText().leng
            errorMessage += "No valid city!\n";
        }

        if (birthdayField.getText() == null || birthdayField.getTex
            errorMessage += "No valid birthday!\n";
        } else {
            if (!DateUtil.validDate(birthdayField.getText())) {
                errorMessage += "No valid birthday. Use the format
            }
        }

        if (errorMessage.length() == 0) {
            return true;
        } else {
            // Show the error message.
            Dialogs.create()
                .title("Invalid Fields")
                .masthead("Please correct invalid fields")
                .message(errorMessage)
                .showError();
            return false;
        }
```

```
    }
}
```

关于该控制器的一些事情应该注意：

1. `setPerson(…)` 方法可以从其它类中调用，用来设置编辑的人员。
2. 当用户点击OK按钮时，调用 `handleOK()` 方法。首先，通过调用 `isInputValid()` 方法做一些验证。只有验证成功，Person对象使用输入的数据填充。这些修改将直接应用到Person对象上，传递给 `setPerson(…)` 。
3. 布尔值 `okClicked` 被使用，以便调用者决定用户是否点击OK或者Cancel按钮。

## 连接视图和控制器

使用已经创建的视图(FXML)和控制器，需要连接到一起。

1. 使用SceneBuilder打开 `PersonEditDialog.fxml` 文件
2. 在左边的*Controller*组中选择 `PersonEditDialogController` 作为控制器类
3. 设置所有TextField的 `fx:id` 到相应的控制器字段上。
4. 设置两个按钮的onAction到相应的处理方法上。

## 打开对话框

在 `MainApp` 中添加一个方法加载和显示编辑人员的对话框。

**MainApp.java**

```
/**
 * Opens a dialog to edit details for the specified person. If the
 * clicks OK, the changes are saved into the provided person object
 * is returned.
 *
 * @param person the person object to be edited
 * @return true if the user clicked OK, false otherwise.
 */
public boolean showPersonEditDialog(Person person) {
    try {
        // Load the fxml file and create a new stage for the popup
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/PersonEd
        AnchorPane page = (AnchorPane) loader.load();

        // Create the dialog Stage.
        Stage dialogStage = new Stage();
        dialogStage.setTitle("Edit Person");
        dialogStage.initModality(Modality.WINDOW_MODAL);
        dialogStage.initOwner(primaryStage);
        Scene scene = new Scene(page);
        dialogStage.setScene(scene);

        // Set the person into the controller.
        PersonEditDialogController controller = loader.getControlle
        controller.setDialogStage(dialogStage);
        controller.setPerson(person);

        // Show the dialog and wait until the user closes it
        dialogStage.showAndWait();

        return controller.isOkClicked();
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}
```

添加下面的方法到 `PersonOverviewController` 中。当用户按下*New*或*Edit*按钮时，这些方法将从 `MainApp` 中调用 `showPersonEditDialog(...)` 。

**PersonOverviewController.java**

```
/**
 * Called when the user clicks the new button. Opens a dialog to ec
 * details for a new person.
 */
@FXML
private void handleNewPerson() {
    Person tempPerson = new Person();
    boolean okClicked = mainApp.showPersonEditDialog(tempPerson);
    if (okClicked) {
        mainApp.getPersonData().add(tempPerson);
    }
}

/**
 * Called when the user clicks the edit button. Opens a dialog to e
 * details for the selected person.
 */
@FXML
private void handleEditPerson() {
    Person selectedPerson = personTable.getSelectionModel().getSele
    if (selectedPerson != null) {
        boolean okClicked = mainApp.showPersonEditDialog(selectedPe
        if (okClicked) {
            showPersonDetails(selectedPerson);
        }

    } else {
        // Nothing selected.
        Dialogs.create()
            .title("No Selection")
            .masthead("No Person Selected")
            .message("Please select a person in the table.")
            .showWarning();
    }
}
```

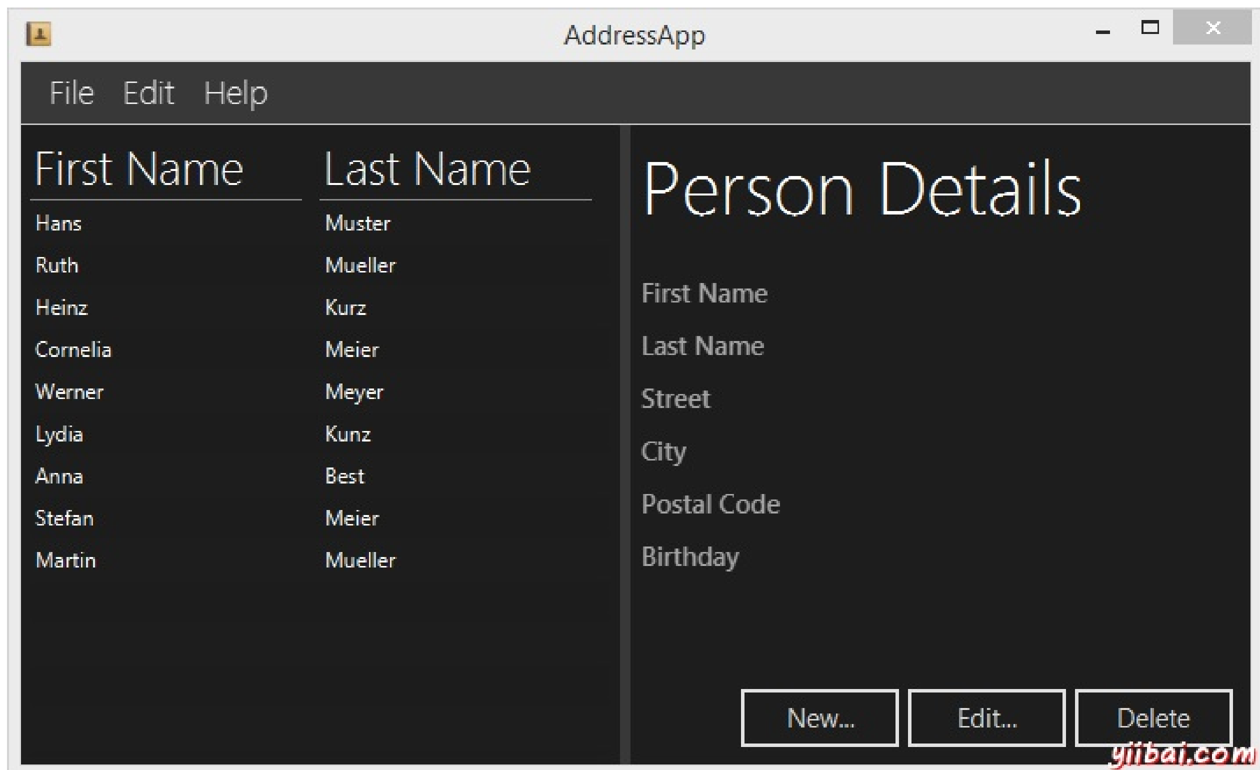
在Scene Builder中打开 `PersonOverview.fxml` 文件，为New和Edit按钮的*On Action*中选择对应的方法。

## 完成!

现在你应该有一个可以工作的*Address*应用。应用能够添加、编辑和删除人员。这里甚至有一些文本字段的验证避免坏的用户输入。

我希望本应用的概念和结构让开始编写自己的JavaFX应用！玩的开心。

# JavaFX - CSS样式 - JavaFX教程




## 第4部分主题

- CSS样式表
- 添加应用程序图标

## CSS样式表

在JavaFX中，你能使用层叠样式表修饰你的用户接口。这非常好！自定义Java应用界面从来不是件简单的事情。

在本教程中，我们将创建一个*DarkTheme*主题，灵感来自于Windows 8 Metro设计。按钮的CSS来至于Pedro Duque Vieia的博客Java中JMetro-Windows 8 Metro控件。

### 熟悉CSS

如果你希望修饰你的JavaFX应用，通常你应该对CSS有一个基本的了解。一个好的起点是CSS教程.

关于CSS更多JavaFX指定信息:

- 使用CSS换肤JavaFX应用 - Oracle教程

- JavaFX CSS参考 - 官方

## 缺省的JavaFX CSS

在JavaFX 8中缺省的CSS风格源码是一个称为 `modena.css` 文件。该CSS文件可以在JavaFX jar文件 `jfxrt.jar` 中找到，它位于Java目录 `/jdk1.8.x/jre/lib/ext/jfxrt.jar` 。

解压 `jfxrt.jar` ，你应该能在 `com/sun/javafx/scene/control/skin/modena/` 目录下找到 `modena.css` 。

缺省的样式表总是应用到JavaFX应用上。通过添加自定义样式表，你能覆盖 `modena.css` 中缺省的样式。

提示：查看缺省的CSS文件能够让你模板你需要覆盖掉那些样式。

## 添加CSS样式表

添加下面的CSS文件 `DarkTheme.css` 到 `view` 包中。

**DarkTheme.css**

```
.background {
    -fx-background-color: #1d1d1d;
}

.label {
    -fx-font-size: 11pt;
    -fx-font-family: "Segoe UI Semibold";
    -fx-text-fill: white;
    -fx-opacity: 0.6;
}

.label-bright {
    -fx-font-size: 11pt;
    -fx-font-family: "Segoe UI Semibold";
    -fx-text-fill: white;
    -fx-opacity: 1;
}

.label-header {
    -fx-font-size: 32pt;
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-opacity: 1;
}

.table-view {
    -fx-base: #1d1d1d;
    -fx-control-inner-background: #1d1d1d;
```

```
    -fx-background-color: #1d1d1d;
    -fx-table-cell-border-color: transparent;
    -fx-table-header-border-color: transparent;
    -fx-padding: 5;
}

.table-view .column-header-background {
    -fx-background-color: transparent;
}

.table-view .column-header, .table-view .filler {
    -fx-size: 35;
    -fx-border-width: 0 0 1 0;
    -fx-background-color: transparent;
    -fx-border-color:
        transparent
        transparent
        derive(-fx-base, 80%)
        transparent;
    -fx-border-insets: 0 10 1 0;
}

.table-view .column-header .label {
    -fx-font-size: 20pt;
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-alignment: center-left;
    -fx-opacity: 1;
}

.table-view:focused .table-row-cell:filled:focused:selected {
    -fx-background-color: -fx-focus-color;
}

.split-pane:horizontal > .split-pane-divider {
    -fx-border-color: transparent #1d1d1d transparent #1d1d1d;
    -fx-background-color: transparent, derive(#1d1d1d,20%);
}

.split-pane {
    -fx-padding: 1 0 0 0;
}

.menu-bar {
    -fx-background-color: derive(#1d1d1d,20%);
}

.context-menu {
    -fx-background-color: derive(#1d1d1d,50%);
}

.menu-bar .label {
    -fx-font-size: 14pt;
```

```
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-opacity: 0.9;
}

.menu .left-container {
    -fx-background-color: black;
}

.text-field {
    -fx-font-size: 12pt;
    -fx-font-family: "Segoe UI Semibold";
}

/*
 * Metro style Push Button
 * Author: Pedro Duque Vieira
 * http://pixelduke.wordpress.com/2012/10/23/jmetro-windows-8-contr
 */
.button {
    -fx-padding: 5 22 5 22;
    -fx-border-color: #e2e2e2;
    -fx-border-width: 2;
    -fx-background-radius: 0;
    -fx-background-color: #1d1d1d;
    -fx-font-family: "Segoe UI", Helvetica, Arial, sans-serif;
    -fx-font-size: 11pt;
    -fx-text-fill: #d8d8d8;
    -fx-background-insets: 0 0 0 0, 0, 1, 2;
}

.button:hover {
    -fx-background-color: #3a3a3a;
}

.button:pressed, .button:default:hover:pressed {
  -fx-background-color: white;
  -fx-text-fill: #1d1d1d;
}

.button:focused {
    -fx-border-color: white, white;
    -fx-border-width: 1, 1;
    -fx-border-style: solid, segments(1, 1);
    -fx-border-radius: 0, 0;
    -fx-border-insets: 1 1 1 1, 0;
}

.button:disabled, .button:default:disabled {
    -fx-opacity: 0.4;
    -fx-background-color: #1d1d1d;
    -fx-text-fill: white;
}
```

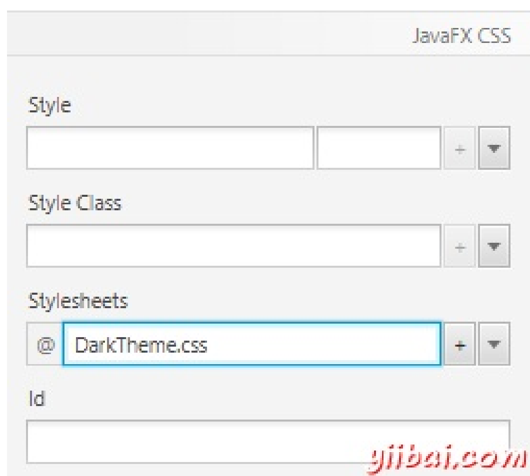
```
.button:default {
    -fx-background-color: -fx-focus-color;
    -fx-text-fill: #ffffff;
}

.button:default:hover {
    -fx-background-color: derive(-fx-focus-color,30%);
}
```

现在我们需要把CSS添加到我们的场景中。我们能在Java代码中编程完成，但是我们将使用SceneBuilder来添加它到fxml文件中。

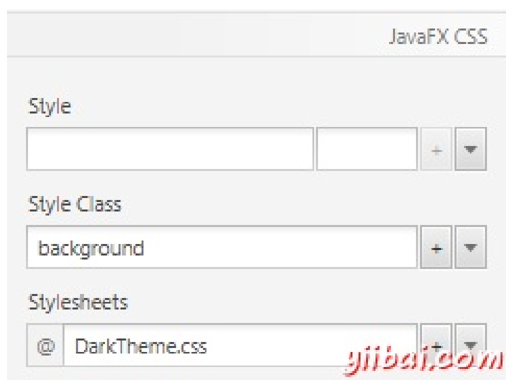
## 添加CSS到RootLayout.fxml

1. 在Scene Builder中打开 `RootLayout.fxml`
2. 在*Hierarchy*视图中选择根节点 `BorderPan` 。在*Properties*组中添加 `DarkTheme.css` 作为样式表。



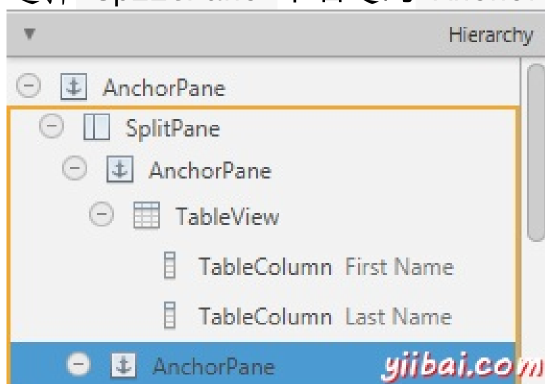

## 添加CSS到PersonEditDialog.fxml

1. 在Scene Builder中打开 `PersonEditDialog.fxml` 。选择根节点 `AnchorPane` ，并且在*Properties*组中选择 `DarkTheme.css` 作为样式表。
2. 背景仍然是白色的，因此添加样式类 `background` 到根节点 `AnchorPane` 。




1. 选择OK按钮，在*Properties*视图中选择*Default Button*单选框。这将修改它的颜色，当用户输入关键词时，使用它作为缺省的按钮。

## 添加CSS到PersonOverview.fxml

1. 在Scene Builder中打开文件 `PersonOverview.fxml` 。在*Hierarchy*组中选择根节点 `AnchorPane` 。在*Properties*下面添加 `DarkTheme.css` 文件作为样式表。
2. 你现在应该已经看到一些修改，表和按钮是黑色的。来自 `modena.css` 中所有类样式 `.table-view` 和 `.button` 应用到表和按钮。因为我们已经在自定义CSS中重定义(因此覆盖掉)一些样式。新的样式自动应用。
3. 你可能需要调整按钮的大小，以便显示所有的文本。
4. 选择 `SplitPane` 中右边的 `AnchorPane` 。


5. 进入到*Properties*组，并且选择 `background` 作为样式表。背景现在应该变为黑色。



## 使用不同样式的标签

现在，在左边的所有的标签都有相同的大小。这里已经有一些样式定义在CSS文件中，称为 `.label-header` 和 `.label-bright` 。我们将使用更多样式的标签Label。

1. 选择*Person Detail*标签，添加 `label-header` 作为样式类。




2. 为了让右边栏的每个Label(显示实际人员的详情)，添加CSS样式类 `label-bright` 。




# 添加应用图标

现在，在标题栏和任务栏中，我们的应用只有一个缺省图标:



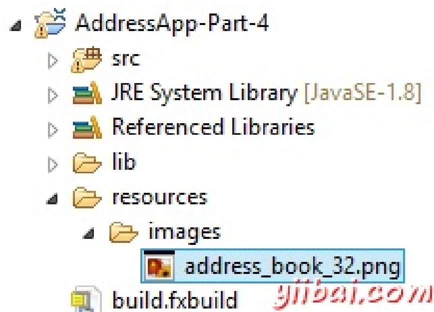

使用自定义图标看起来更好了。




## 图标文件

获取图标的一个可能地方是Icon Finder。我下载了一个地址本的图标.

通常在你的AddressApp项目中创建一个目录称为resources，在它中子目录称为images。把你选择的图标放入到images目录中。现在，你的目录结构应该看上去如下所示：




## 设置图标到场景

为了给你场景设置图标，添加下面一行到 `MainApp.jar` 的 `start(…)` 方法中。

### MainApp.java

```
this.primaryStage.getIcons().add(new Image("file:resources/images/a
```

现在，整个 `start(…)` 方法看上去应该是这样的。：

```
public void start(Stage primaryStage) {
    this.primaryStage = primaryStage;
    this.primaryStage.setTitle("AddressApp");

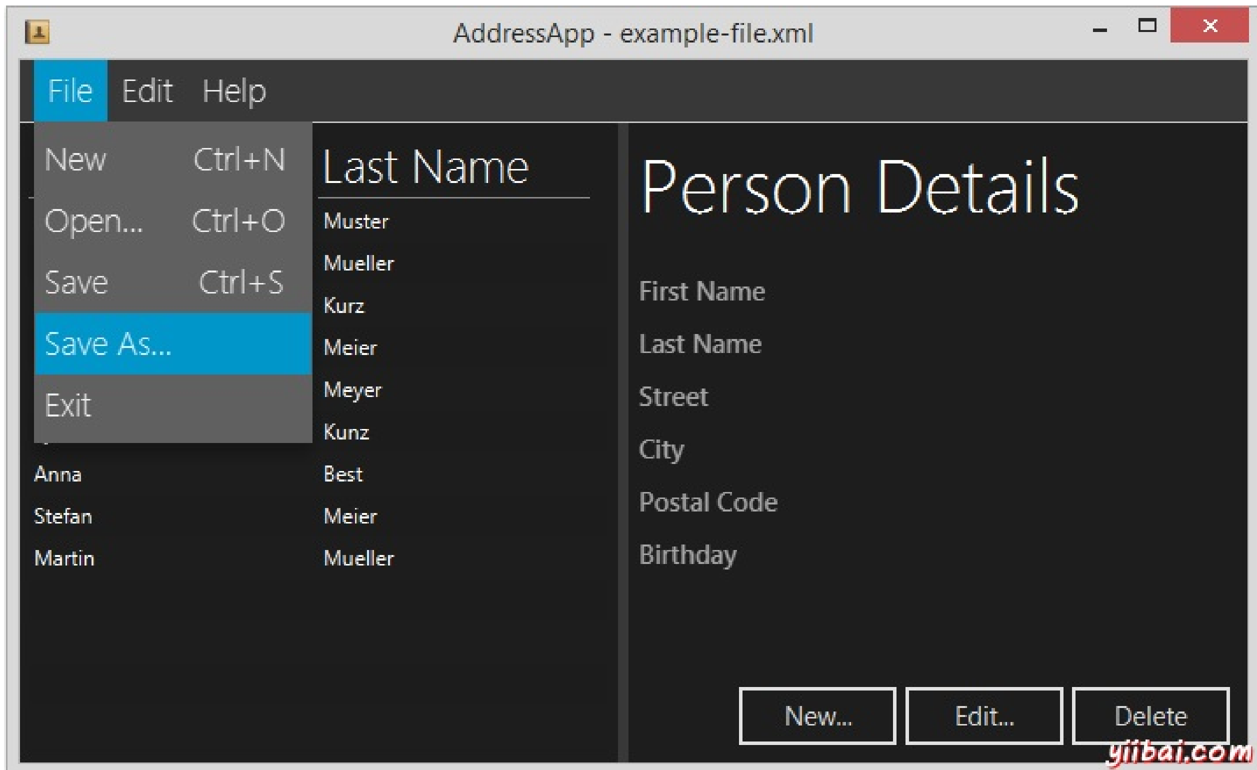
    // Set the application icon.
    this.primaryStage.getIcons().add(new Image("file:resources/imag

    initRootLayout();

    showPersonOverview();
}
```

当然，你也应该添加图标到人员编辑对话框的Stage中。

# JavaFX - XML格式存储 - JavaFX教程




## 第5部分的主题

- 持久化数据为XML
- 使用JavaFX的FileChooser
- 使用JavaFX的菜单
- 在用户设置中保存最后打开的文件路径。

现在我们的地址应用程序的数据只保存在内存中。每次我们关闭应用程序，数据将丢失，因此是时候开始考虑持久化存储数据了。

## 保存用户设置

Java允许我们使用 `Preferences` 类保存一些应用状态。依赖于操作系统， `Perferences` 保存在不同的地方(例如：Windows中的注册文件)。

我们不能使用 `Preferences` 来保存全部地址簿。但是它允许我们保存一些简单的应用状态。一件这样事情是最后打开文件的路径。使用这个信息，我们能加载最后应用的状态，不管用户什么时候重启应用程序。

下面两个方法用于保存和检索Preference。添加它们到你的 `MainApp` 类的最后：

**MainApp.java**

```
/**
 * Returns the person file preference, i.e. the file that was last
 * The preference is read from the OS specific registry. If no such
 * preference can be found, null is returned.
 *
 * @return
 */
public File getPersonFilePath() {
    Preferences prefs = Preferences.userNodeForPackage(MainApp.clas
    String filePath = prefs.get("filePath", null);
    if (filePath != null) {
        return new File(filePath);
    } else {
        return null;
    }
}

/**
 * Sets the file path of the currently loaded file. The path is per
 * the OS specific registry.
 *
 * @param file the file or null to remove the path
 */
public void setPersonFilePath(File file) {
    Preferences prefs = Preferences.userNodeForPackage(MainApp.clas
    if (file != null) {
        prefs.put("filePath", file.getPath());

        // Update the stage title.
        primaryStage.setTitle("AddressApp - " + file.getName());
    } else {
        prefs.remove("filePath");

        // Update the stage title.
        primaryStage.setTitle("AddressApp");
    }
}
```

# 持久性数据到XML

## 为什么是XML？

持久性数据的一种最常用的方法是使用数据库。数据库通常包含一些类型的关系数据(例如：表)，当我们需要保存的数据是对象时。这称object-relational impedance mismatch。匹配对象到关系型数据库表有很多工作要做。这里有一些框架帮助我们匹配(例如：Hibernate，最流行的一个)。但是它仍然需要相当多的设置工作。

对于简单的数据模型，非常容易使用XML。我们使用称为JAXB(Java Architecture for XML Binding)的库。只需要几行代码，JAXB将允许我们生成XML输出，如下所示：

示例**XML**输出

```
<persons>
    <person>
        <birthday>1999-02-21</birthday>
        <city>some city</city>
        <firstName>Hans</firstName>
        <lastName>Muster</lastName>
        <postalCode>1234</postalCode>
        <street>some street</street>
    </person>
    <person>
        <birthday>1999-02-21</birthday>
        <city>some city</city>
        <firstName>Anna</firstName>
        <lastName>Best</lastName>
        <postalCode>1234</postalCode>
        <street>some street</street>
    </person>
</persons>
```

## 使用**JAXB**

JAXB已经包含在JDK中。这意味着我们不需要包含任何其它的库。

JAXB提供两个主要特征：编列(marshal)Java对象到XML的能力，反编列(unmarshal)XML到Java对象。

为了让JAXB能够做转换，我们需要准备我们的模型。

## 准备**JAXB**的模型类

我们希望保持的数据位于 `MainApp` 类的 `personData` 变量中。JAXB要求使用 `@XmlRootElement` 注释作为最顶层的类。 `personData` 是 `ObservableList` 类，我们不能把任何注释放到 `ObservableList` 上。因此，我们需要创建另外一个类，它只用于保存 `Person` 列表，用于存储成XML文件。

创建的新类名为 `PersonListWrapper` ，把它放入到 `ch.makery.address.model` 包中。

**PersonListWrapper.java**

```
package ch.makery.address.model;

import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * Helper class to wrap a list of persons. This is used for saving
 * list of persons to XML.
 *
 * @author Marco Jakob
 */
@XmlRootElement(name = "persons")
public class PersonListWrapper {

    private List<Person> persons;

    @XmlElement(name = "person")
    public List<Person> getPersons() {
        return persons;
    }

    public void setPersons(List<Person> persons) {
        this.persons = persons;
    }
}
```

注意两个注释：

- `@XmlRootElement` 定义根元素的名称。
- `@XmlElement` 一个可选的名称，用来指定元素。

## 使用JAXB读写数据

我们让 `MainApp` 类负责读写人员数据。添加下面两个方法到 `MainApp.java` 的最后：

```
/**
 * Loads person data from the specified file. The current person da
 * be replaced.
 *
 * @param file
 */
public void loadPersonDataFromFile(File file) {
    try {
        JAXBContext context = JAXBContext
                .newInstance(PersonListWrapper.class);
        Unmarshaller um = context.createUnmarshaller();
```

```
        // Reading XML from the file and unmarshalling.
        PersonListWrapper wrapper = (PersonListWrapper) um.unmarsha

        personData.clear();
        personData.addAll(wrapper.getPersons());

        // Save the file path to the registry.
        setPersonFilePath(file);

    } catch (Exception e) { // catches ANY exception
        Dialogs.create()
                .title("Error")
                .masthead("Could not load data from file:\n" + file
                .showException(e);
    }
}

/**
 * Saves the current person data to the specified file.
 *
 * @param file
 */
public void savePersonDataToFile(File file) {
    try {
        JAXBContext context = JAXBContext
                .newInstance(PersonListWrapper.class);
        Marshaller m = context.createMarshaller();
        m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

        // Wrapping our person data.
        PersonListWrapper wrapper = new PersonListWrapper();
        wrapper.setPersons(personData);

        // Marshalling and saving XML to the file.
        m.marshal(wrapper, file);

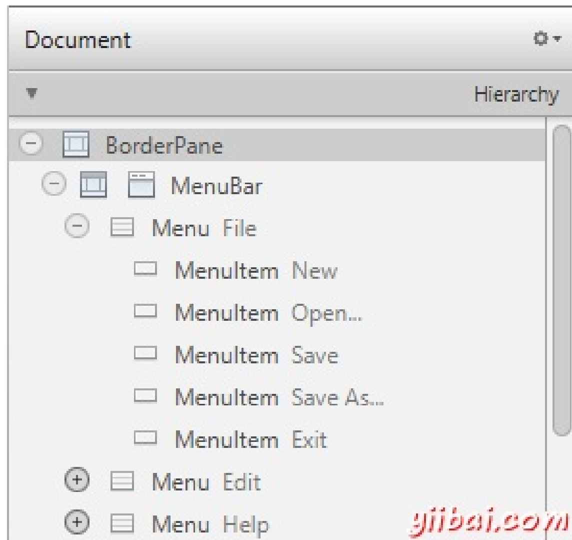
        // Save the file path to the registry.
        setPersonFilePath(file);
    } catch (Exception e) { // catches ANY exception
        Dialogs.create().title("Error")
                .masthead("Could not save data to file:\n" + file.g
                .showException(e);
    }
}
```

编组和解组已经准备好，让我们创建保存和加载的菜单实际的使用它。

## 处理菜单响应

在我们 `RootLayout.fxml` 中，这里已经有一个菜单，但是我们没有使用它。在我们添加响应到菜单中之前，我们首先创建所有的菜单项。

在Scene Builder中打开 `RootLayout.fxml` ，从*library*组中拖曳必要的菜单到*Hierarchy*组的 `MemuBar` 中。创建New，Open...，Save，Save As...和Exit菜单项。




提示：使用*Properties*组下的*Accelerator*设置，你能设置菜单项的快捷键。

## RootLayoutController

为了处理菜单动作，我们需要创建一个新的控制器类。在控制器包 `ch.makery.address.view` 中创建一个类 `RootLayoutController` 。

添加下面的内容到控制器中：

### RootLayoutController.java

```
package ch.makery.address.view;

import java.io.File;

import javafx.fxml.FXML;
import javafx.stage.FileChooser;

import org.controlsfx.dialog.Dialogs;

import ch.makery.address.MainApp;

/**
 * The controller for the root layout. The root layout provides the
 * application layout containing a menu bar and space where other J
 * elements can be placed.
 *
 * @author Marco Jakob
```

```
 */
public class RootLayoutController {

    // Reference to the main application
    private MainApp mainApp;

    /**
     * Is called by the main application to give a reference back t
     *
     * @param mainApp
     */
    public void setMainApp(MainApp mainApp) {
        this.mainApp = mainApp;
    }

    /**
     * Creates an empty address book.
     */
    @FXML
    private void handleNew() {
        mainApp.getPersonData().clear();
        mainApp.setPersonFilePath(null);
    }

    /**
     * Opens a FileChooser to let the user select an address book t
     */
    @FXML
    private void handleOpen() {
        FileChooser fileChooser = new FileChooser();

        // Set extension filter
        FileChooser.ExtensionFilter extFilter = new FileChooser.Ext
                "XML files (*.xml)", "*.xml");
        fileChooser.getExtensionFilters().add(extFilter);

        // Show save file dialog
        File file = fileChooser.showOpenDialog(mainApp.getPrimarySt

        if (file != null) {
            mainApp.loadPersonDataFromFile(file);
        }
    }

    /**
     * Saves the file to the person file that is currently open. If
     * open file, the "save as" dialog is shown.
     */
    @FXML
    private void handleSave() {
        File personFile = mainApp.getPersonFilePath();
        if (personFile != null) {
            mainApp.savePersonDataToFile(personFile);
```

```
        } else {
            handleSaveAs();
        }
    }

    /**
     * Opens a FileChooser to let the user select a file to save to
     */
    @FXML
    private void handleSaveAs() {
        FileChooser fileChooser = new FileChooser();

        // Set extension filter
        FileChooser.ExtensionFilter extFilter = new FileChooser.Ext
                "XML files (*.xml)", "*.xml");
        fileChooser.getExtensionFilters().add(extFilter);

        // Show save file dialog
        File file = fileChooser.showSaveDialog(mainApp.getPrimarySt

        if (file != null) {
            // Make sure it has the correct extension
            if (!file.getPath().endsWith(".xml")) {
                file = new File(file.getPath() + ".xml");
            }
            mainApp.savePersonDataToFile(file);
        }
    }

    /**
     * Opens an about dialog.
     */
    @FXML
    private void handleAbout() {
        Dialogs.create()
            .title("AddressApp")
            .masthead("About")
            .message("Author: Marco Jakob\nWebsite: http://code.mak
            .showInformation();
    }

    /**
     * Closes the application.
     */
    @FXML
    private void handleExit() {
        System.exit(0);
    }
}
```

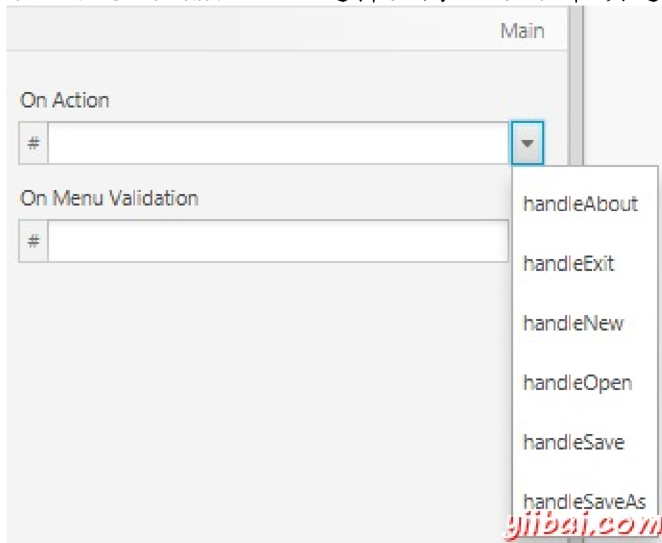
## FileChooser

注意在上面的 `RootLayoutController` 中使用 `FileCooser` 的方法。首先，创建新的 `FileChooser` 类对象的，然后，添加扩展名过滤器，以至于只显示以 `.xml` 结尾的文件。最后，文件选择器显示在主Stage的上面。

如果用户没有选择一个文件关闭对话框，返回 `null` 。否则，我们获得选择的文件，我们能传递它到 `MainApp` 的 `loadPersonDataFromFile(…)` 或 `savePersonDataToFile()` 方法中。

## 连接fxml视图到控制器

1. 在Scene Builder中打开 `RootLayout.fxml` 。在*Controller*组中选择 `RootLayoutController` 作为控制器类。
2. 回到*Hierarchy*组中，选择一个菜单项。在*Code*组中On Action下，应该看到所有可用控制器方法的选择。为每个菜单项选择响应的方法。




3. 为每个菜单项重复第2步。
4. 关闭Scene Builder，并且在项目的根目录上按下刷新F5。这让Eclipse知道在Scene Builder中所做的修改。

## 连接MainApp和RootLayoutController

在几个地方， `RootLayoutController` 需要引用 `MainApp` 类。我们也没有传递一个 `MainApp` 的引用到 `RootLayoutController` 。

打开 `MainApp` 类，使用下面的替代 `initRootLayout()` 方法：

```
/**
 * Initializes the root layout and tries to load the last opened
 * person file.
 */
public void initRootLayout() {
    try {
        // Load root layout from fxml file.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class
                .getResource("view/RootLayout.fxml"));
        rootLayout = (BorderPane) loader.load();

        // Show the scene containing the root layout.
        Scene scene = new Scene(rootLayout);
        primaryStage.setScene(scene);

        // Give the controller access to the main app.
        RootLayoutController controller = loader.getController();
        controller.setMainApp(this);

        primaryStage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }

    // Try to load last opened person file.
    File file = getPersonFilePath();
    if (file != null) {
        loadPersonDataFromFile(file);
    }
}
```

注意两个修改：一行给控制器访问*MainApp*和最后三行加载最新打开的人员文件。

## 测试

做应用程序的测试驱动，你应该能够使用菜单保存人员数据到文件中。

当你在编辑器中打开一个 `xml` 文件，你将注意到生日没有正确保存，这是一个空的 `<birthday/>` 标签。原因是JAXB不只奥如何转换 `LocalDate` 到XML。我们必须提供一个自定义的 `LocalDateAdapter` 定义这个转换。

在 `ch.makery.address.util` 中创建新的类，称为 `LocalDateAdapter` ，内容如下：

**LocalDateAdapter.java**

```
package ch.makery.address.util;

import java.time.LocalDate;

import javax.xml.bind.annotation.adapters.XmlAdapter;

/**
 * Adapter (for JAXB) to convert between the LocalDate and the ISO
 * String representation of the date such as '2012-12-03'.
 *
 * @author Marco Jakob
 */
public class LocalDateAdapter extends XmlAdapter<String, LocalDate>

    @Override
    public LocalDate unmarshal(String v) throws Exception {
        return LocalDate.parse(v);
    }

    @Override
    public String marshal(LocalDate v) throws Exception {
        return v.toString();
    }
}
```

然后打开 `Person.jar` ，添加下面的注释到 `getBirthday()` 方法上：

```
@XmlJavaTypeAdapter(LocalDateAdapter.class)
public LocalDate getBirthday() {
    return birthday.get();
}
```

现在，再次测试。试着保存和加载XML文件。在重启之后，它应该自动加载最后使用的文件。

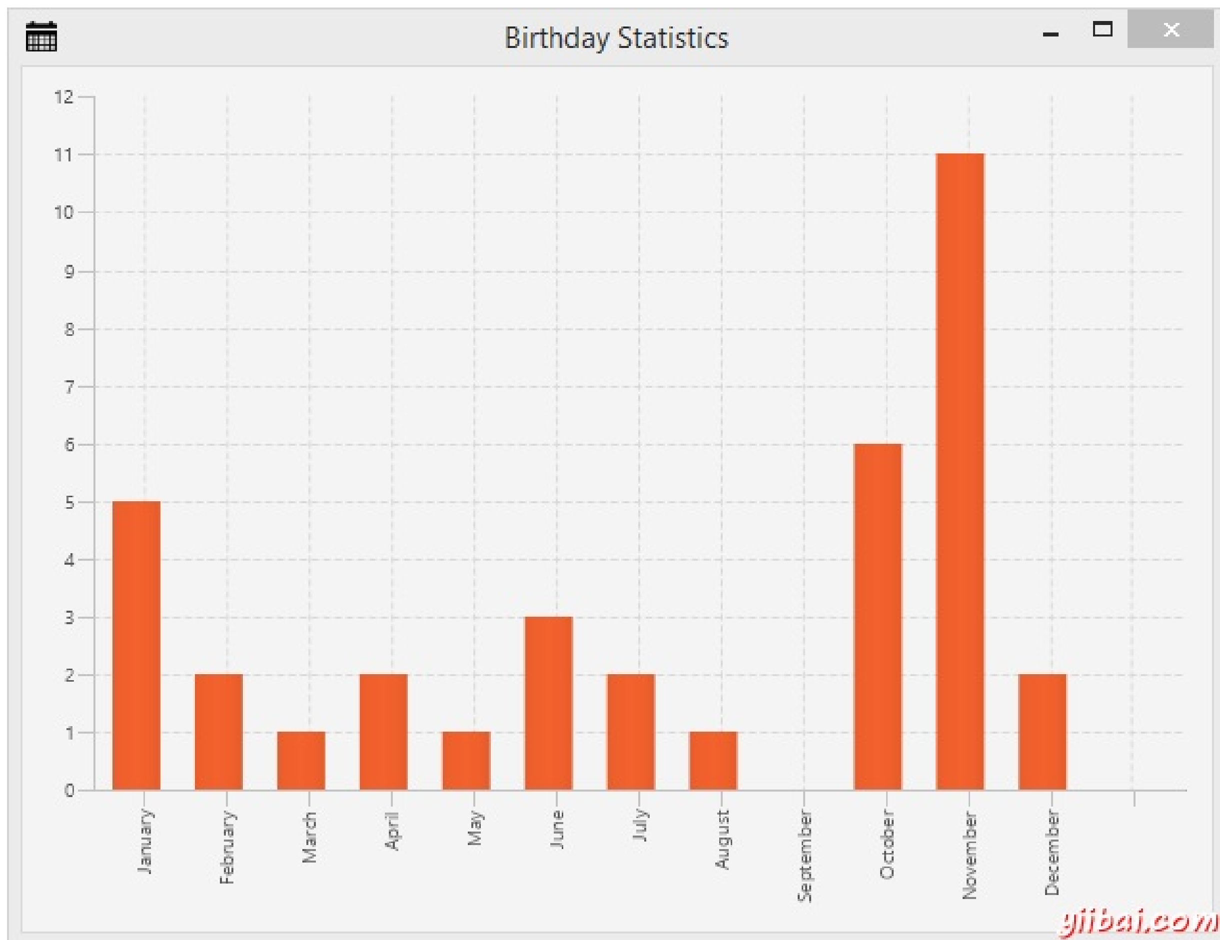
# 它如何工作

让我们看下它是如何一起工作的：

1. 应用程序使用 `MainApp` 中的 `main(…)` 方法启动。
2. 调用 `public MainApp()` 构造函数添加一些样例数据。
3. 调用 `MainApp` 的 `start(…)` 方法，调用 `initRootLayout()` 从 `RootLayout.fxml` 中初始化根布局。fxml文件有关于使用控制器的信息，连接视图到 `RootLayoutController` 。
4. `MainApp` 从fxml加载器中获取 `RootLayoutController` ，传递自己的引用到控制器中。使用这些引用，控制器随后可以访问 `MainApp` 的公开方法。

5. 在 `initRootLayout` 方法结束，我们试着从 `Perferences` 中获取最后打开的人员文件。如果 `Perferences` 知道有这样一个XML文件，我们将从这个XML文件中加载数据。这显然会覆盖掉构造函数中的样例数据。

# JavaFX - 统计图 - JavaFX教程




## 第6部分的主题

- 创建一个统计图显示生日的分布。

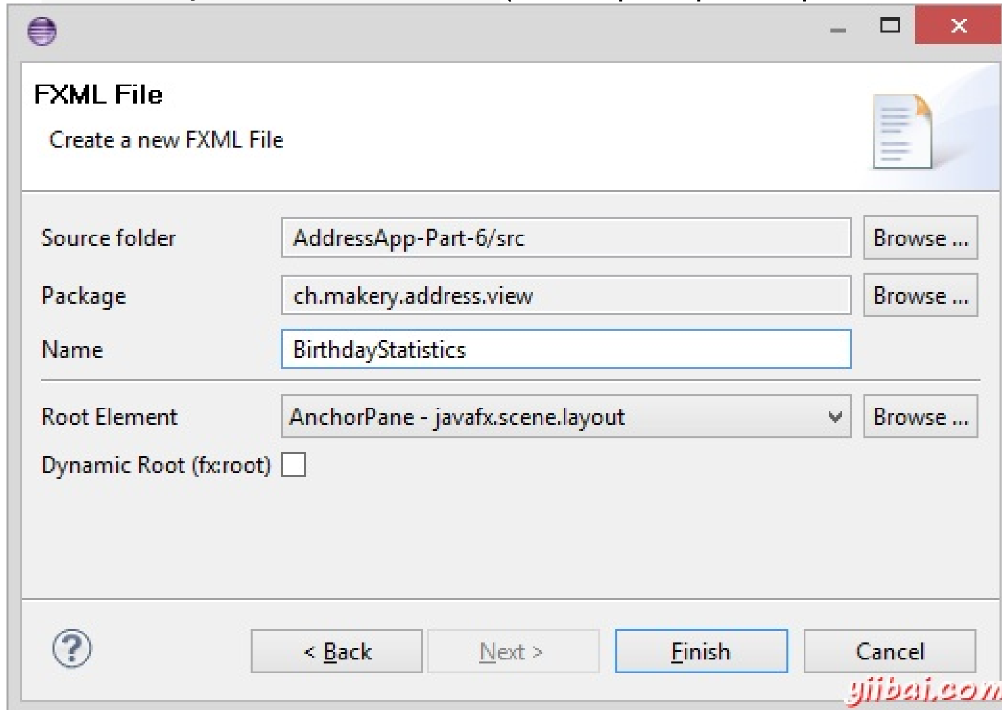
## 生日统计

在AddressApp中所有人员都有生日。当我们人员庆祝他们生日的时候，如果有一些生日的统计不是会更好。

我们使用柱状图，包含每个月的一个条形。每个条形显示在指定月份中有多少人需要过生日。

## 统计FXML视图

1. 在 `ch.makery.address.view` 包中我们开始创建一个 `BirthdayStatistics.fxml` (右击包|*New*|*other..*|*New FXML Document*)




2. 在Scene Builder中打开 `BirthdayStatistics.fxml` 文件。
3. 选择根节点 `AnchorPane` 。在*Layout*组中设置*Pref Width*为620，*Pref Height*为450。
4. 添加 `BarChart` 到 `AnchorPane` 中。
5. 右击 `BarChart` 并且选择*Fit to Parent*。
6. 保存fxml文件，进入到Eclipse中，F5刷新项目。

在我们返回到Scene Builder之前，我们首先创建控制器，并且在我们的 `MainApp` 中准备好一切。

# 统计控制器

在view包 `ch.makery.address.view` 中创建一个Java类，称为 `BirthdayStatisticsController.java` 。

在开始解释之前，让我们看下整个控制器类。

**BirthdayStatisticsController.java**

```
package ch.makery.address.view;

import java.text.DateFormatSymbols;
import java.util.Arrays;
import java.util.List;
```

```
import java.util.Locale;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.XYChart;
import ch.makery.address.model.Person;

/**
 * The controller for the birthday statistics view.
 *
 * @author Marco Jakob
 */
public class BirthdayStatisticsController {

    @FXML
    private BarChart<String, Integer> barChart;

    @FXML
    private CategoryAxis xAxis;

    private ObservableList<String> monthNames = FXCollections.obse

    /**
     * Initializes the controller class. This method is automatical
     * after the fxml file has been loaded.
     */
    @FXML
    private void initialize() {
        // Get an array with the English month names.
        String[] months = DateFormatSymbols.getInstance(Locale.ENGL
        // Convert it to a list and add it to our ObservableList of
        monthNames.addAll(Arrays.asList(months));

        // Assign the month names as categories for the horizontal
        xAxis.setCategories(monthNames);
    }

    /**
     * Sets the persons to show the statistics for.
     *
     * @param persons
     */
    public void setPersonData(List<Person> persons) {
        // Count the number of people having their birthday in a sp
        int[] monthCounter = new int[12];
        for (Person p : persons) {
            int month = p.getBirthday().getMonthValue() - 1;
            monthCounter[month]++;
        }
```

```
        XYChart.Series<String, Integer> series = new XYChart.Series

        // Create a XYChart.Data object for each month. Add it to t
        for (int i = 0; i < monthCounter.length; i++) {
            series.getData().add(new XYChart.Data<>(monthNames.get(
        }

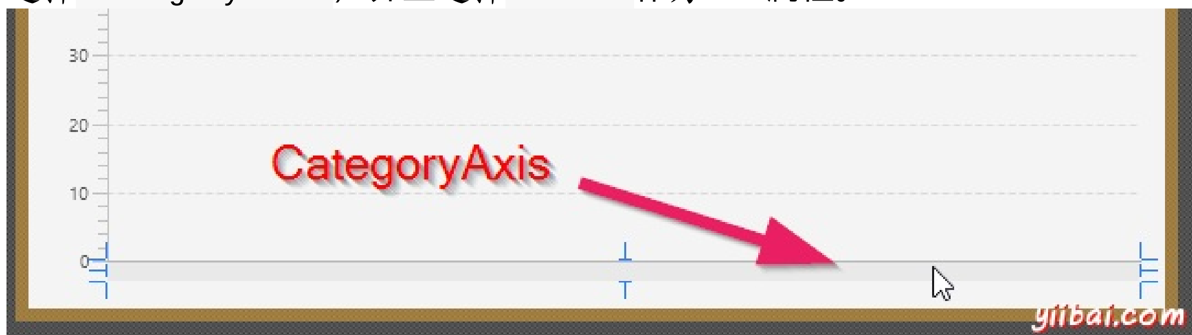
        barChart.getData().add(series);
    }
}
```

## 控制器如何工作

1. 控制器需要从FXML文件中访问两个元素:
    - `barChar` ：它有 `String` 和 `Integer` 类型。 `String` 用于x轴上的月份， `Integer` 用于指定月份中人员的数量。
    - `xAxis` ：我们使用它添加月字符串
2. `initialize()` 方法使用所有月的列表填充 `x-axis` 。
3. `setPersonData(…)` 方法将由 `MainApp` 访问，设置人员数据。它遍历所有人员，统计出每个月生日的人数。然后它为每个月添加 `XYChart.Data` 到数据序列中。每个 `XYChart.Data` 对象在图表中表示一个条形。

## 连接视图和控制器

1. 在Scene Builder中打开 `BirthdayStatistics.fxml` 。
2. 在Controller组中设置 `BirthdayStatisticsController` 为控制器。
3. 选择 `BarChart` ，并且选择 `barChar` 作为fx:id属性(在*Code*组中)
4. 选择 `CategoryAxis` ，并且选择 `xAxis` 作为fx:id属性。




5. 你可以添加一个标题给 `BarChar` (在*Properties*组中)进一步修饰。

## 连接View/Controller和MainApp

我们为生日统计使用与编辑人员对话框相同的机制，一个简单的弹出对话框。

添加下面的方法到 `MainApp` 类中

```
/**
 * Opens a dialog to show birthday statistics.
 */
public void showBirthdayStatistics() {
    try {
        // Load the fxml file and create a new stage for the popup.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/Birthday
        AnchorPane page = (AnchorPane) loader.load();
        Stage dialogStage = new Stage();
        dialogStage.setTitle("Birthday Statistics");
        dialogStage.initModality(Modality.WINDOW_MODAL);
        dialogStage.initOwner(primaryStage);
        Scene scene = new Scene(page);
        dialogStage.setScene(scene);

        // Set the persons into the controller.
        BirthdayStatisticsController controller = loader.getControl
        controller.setPersonData(personData);

        dialogStage.show();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

一切设置完毕，但是我们没有任何东西实际上调用新的 `showBirthdayStatistics()` 方法。幸运的是我们已经在 `RootLayout.fxml` 中有一个菜单，它可以用于这个目的。

## 显示生日统计菜单

在 `RootLayoutController` 中添加下面的方法，它将处理显示生日统计菜单项的用户点击。

```
/**
 * Opens the birthday statistics.
 */
@FXML
private void handleShowBirthdayStatistics() {
  mainApp.showBirthdayStatistics();
}
```

现在，使用Scene Builder打开 `RootLayout.fxml` 文件。创建*Staticstic* `菜单`，带有一个*Show Statistcs* `MenuItem`：




选择*Show Statistics* `MenuItem`，并且选择 `handleShowBirthdayStatistics` 作为 `On Action` (在*Code*组中)。

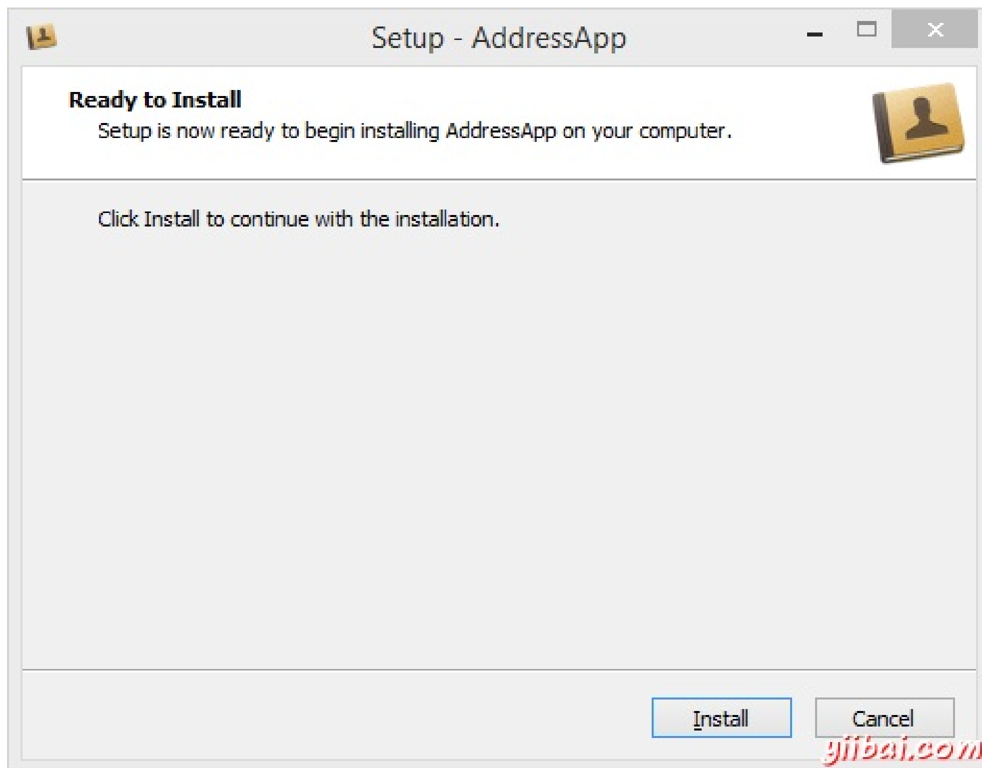



进入到Eclipse，刷新项目，测试它。

# JavaFX Chart的更多信息

更多信息的一个好地方是官方Oracle教程，使用JavaFX Chart.

# JavaFX - 部署 - JavaFX教程




我想已经写到本教程系列的最后一部分了，应该教你如何部署(例如：打包和发布)AddressApp

## 第7部分的主题

- 使用e(fx)clipse本地包(Native Package)部署我们的JavaFX应用程序。

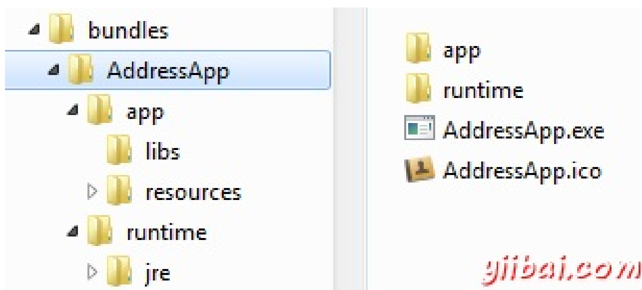
## 什么是部署

部署是打包和发布软件给用户的过程。这是软件开发的关键部分，因为它是第一次与使用我们软件的用户交流。

Java的广告口号是编写一次，到处运行，这说明Java语言的跨平台好处。理想情况下，这意味着我们Java应用可以运行在任何装备有JVM的设备上。 在过去，安装Java应用程序的用户经验不总是平滑的。如果用户在系统中没有要求的Java版本，它必须首先直接安装它。这导致有些困难，例如，需要管理员权限，Java版本之间的兼容问题等等。 幸运的是，JavaFX提供新的部署选项称为本地打包(也称为自包含应用程序包)。一个本地包是一个包含你的应用代码和平台指定的Java运行时的打包程序。 Oracle提供的官方JavaFx文档包含一个所有JavaFX部署选项的扩展指南。 在本章节中，我们教你如何使用Eclipse和e(fx)clipse插件创建本地包。

# 创建本地包

目标是在用户的计算机上单个目录中创建一个自包含的应用程序。下面是AddressApp应用看起来的样子(在Windows上):




`app` 目录包含我们的应用数据和 `runtime` 目录(包含平台相关的Java运行时)。

为了让用户更加舒适，我们也提供一个安装器：

- Windows下的 `exe` 文件安装器
- MacOS下的 `dmg` (拖放)安装器。

E(fx)clipse插件会帮助我们生成本地包和安装器。

## 第1步 编辑build.fxbuild

E(fx)clipse使用 `build.fxbuild` 文件生成一个被Ant编译工具使用的文件。(如果你没有一个 `build.fxbuid` 文件，在Eclipse中创建一个新的Java FX项目，并且拷贝生成的文件过来。

1. 从项目的根目录下打开 `build.fxbuild` 。

2. 填写包含一个星号的字段。对于*MacOS*：在应用程序标题中不能使用空格，因为好像会产生问题。

FX Build Configuration

Build & Package Properties

The following properties are needed to build the JavaFX-Application

Build Directory*: ${project}/build Filesystem ... Workspace ...

Vendor name*: makery.ch

Application title*: AddressApp

Application version*: 1.0

Application class*: ch.makery.address.MainApp Browse ...

Preloader class: Browse ...

Splash: Browse ...

Manifest-Attributes: Name Value Add ... Remove

Toolkit Type:

Packaging Format: exe

automatic Proxy Resolution

Convert CSS into binary form

Enable verbose build mode (Not recommended)

Building Exporting

To generate build instructions and export the project:

- Generate ant build.xml only
- Generate ant build.xml and run

yiibai.com

3. 在Windows下Packaging Format选择 `exe` ，MacOS下选择 `dmg` ，Linux下选择 `rpm`
4. 点击 `Generate ant build.xml only` 的连接(在右边可以找到)。

Building Exporting

To generate build instructions and export the project:

- Generate ant build.xml only
- Generate ant build.xml and run

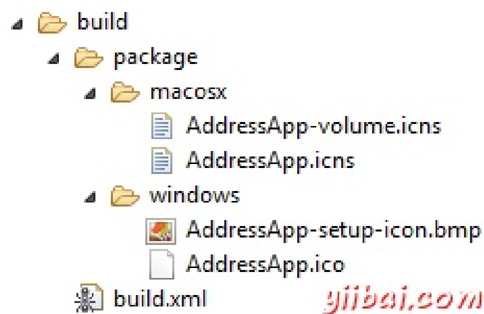
5. 验证是否创建一个新的 `build` 目录和文件 `build.xml`

## 第2步 添加安装程序的图标

我们希望安装程序有一些好看的图标:

- AddressApp.ico 安装文件图标
- AddressApp-setup-icon.bmp 安装启动画面图标
- AddressApp.icns Mac安装程序图标

- 在 `build` 目录下创建下面的子目录:
  - `build/package/windows` (只用于Windows)
  - `build/package/macos` (只用于macos)
- 拷贝上面的相关图标到这些目录中，现在它应该看起来如下所示:

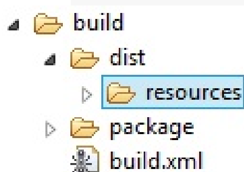



- 重要：图标的名称必须精确匹配 `build.fxbuild` 中指定的Application的标题名：
    - `YourAppTitle.ico`
    - `YourAppTitle-setup-icon.bmp`
    - `YourAppTitle.icns`

## 第3步 添加资源

我们的 `resources` 目录不能自动拷贝。我们必须手动添加它到build目录下:

1. 在 `build` 目录下创建下面的子目录:
    - `build/dist`
2. 拷贝 `resources` 目录(包含我们应用的图标)到 `build/dist` .




## 第4步 编辑build.xml包含图标

E(fx)clipse生成的 `build/build.xml` 文件(准备使用Ant执行)。我们的安装器图标和资源图像不能正常工作。

当e(fx)clipse没有告诉它包含其它资源，例如 `resources` 目录和上面添加的安装文件图标时，我们必须手动编辑 `build.xml` 文件。

打开 `build.xml` 文件，找到路径 `fxant` 。添加一行到 `${basedir}` (将让我们安装器图标可用)。

**build.xml - 添加"basedir"**

```
<path id="fxant">
  <filelist>
    <file name="${java.home}\..\lib\ant-javafx.jar"/>
    <file name="${java.home}\lib\jfxrt.jar"/>
    <file name="${basedir}"/>
  </filelist>
</path>
```

找到块 `fx:resources id="appRes"` ，文件的更下面位置。为 `resources` 添加一行：

**build.xml - 添加"resources"**

```
<fx:resources id="appRes">
    <fx:fileset dir="dist" includes="AddressApp.jar"/>
    <fx:fileset dir="dist" includes="libs/*"/>
    <fx:fileset dir="dist" includes="resources/**"/>
</fx:resources>
```

有时候，版本数不能添加到 `fx:application` 中，使得安装器总是缺省的版本 `1.0` (在注释中很多人指出这个问题)。为了修复它，手动添加版本号(感谢Marc找到解决办法)。 解决):

**build.xml - 添加 "version"**

```
<fx:application id="fxApplication"
    name="AddressApp"
    mainClass="ch.makery.address.MainApp"
    version="1.0"
/>
```

现在，我们已经能够使用ant编译运行 `build.xml` 了。这将会生成一个可运行的项目jar文件。但是我们希望更进一步， 创建一个很好的安装器。

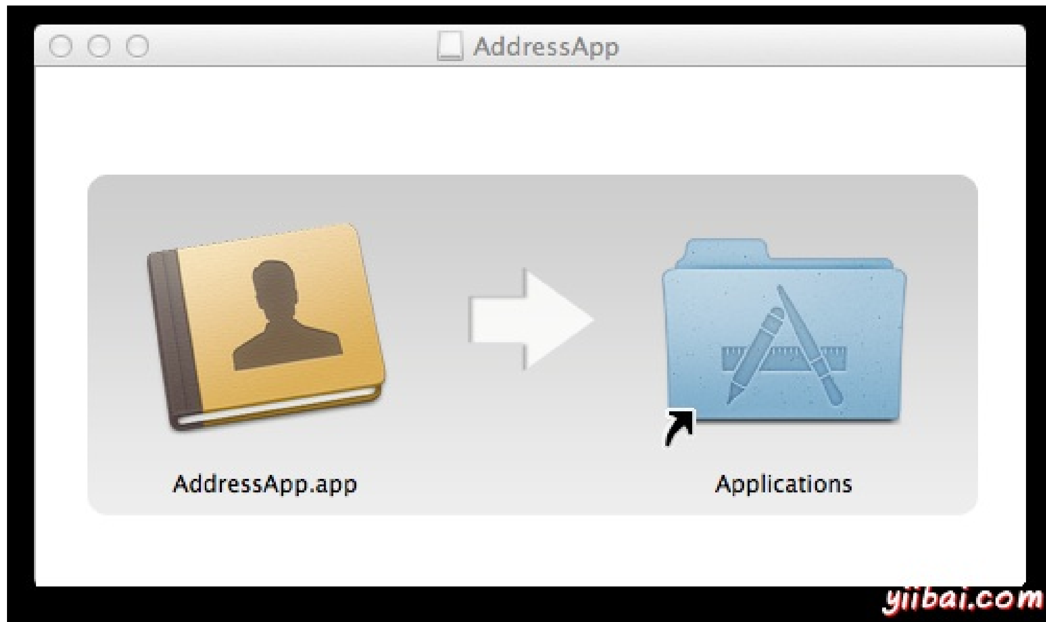
## 第5步(Windows) - Windows exe安装器




使用Inno Setup，我们能为我们的应用程序创建一个单独 `.exe` 文件的Windows安装器。生成的 `.exe` 执行用户级别的安装(无需管理员权限)。也创建一个快捷方式(菜单和桌面)。

1. 下载Inno Setup 5以后版本，安装Inno程序到你的计算机上。我们的Ant脚本将使用它自动生成安装器。
2. 告诉Windows Inno程序的安装路径(例如： `C:\Program Files (x86)\Inno Setup 5)` 。添加Inno安装路径到Path环境变量中。如果你不知道哪里可以找到它，阅读Windows中如何设置路径和环境变量.
3. 重启Eclipse，并且继续第6步。

## 第5步(MAC) - MacOS dmg安装器




为了创建Mac OS `dmg` 拖放安装器，不需要任何的要求。

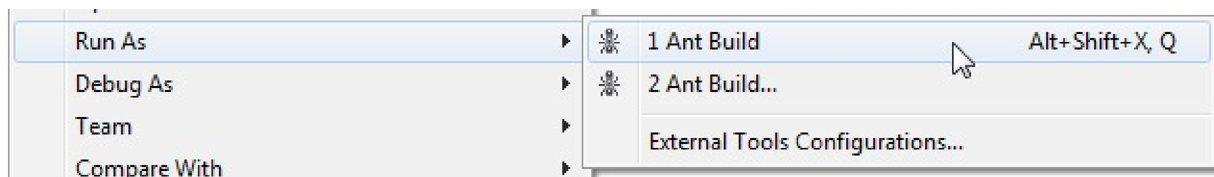
注意：为了让安装器映像能工作，它的名称必须与应用名称相同。

## 第5步(Linux等) Linux rpm安装器

其它打包选项(Windows的 `msi` ，Linux的 `rpm` )参考本地打包博客 或者本oracle文档.

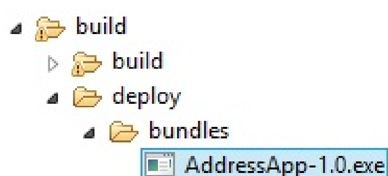
## 第6步 运行build.xml

最后一步，我们使用ant运行 `build.xml` ，右击 `build.xml` 文件| *Run As* | *Ant Build*。




编译将运行一会(在我的计算机上大概1分钟)。

如果一切都成功，你应该在 `build/deploy/bundles` 目录下找到本地打包。Windows版本看起来如下所示：



文件 `AddressApp-1.0.exe` 可以用作为单个文件安装应用。该安装程序将拷贝打包到 `C:/Users/[yourname]/AppData/Local/AddressApp` 目录下。

# Java.io包教程



Java.io包提供了用于系统的输入和输出，通过数据流，序列化和文件系统。本参考将引导您完成java.io包中提供简单，实用的方法和实例。

## 读者

该参考是为初学者而准备，帮助他们了解相关Java.io包中所有可用的方法的基本功能。

## 必备条件

在开始做练习的各类在此引用给定的例子，假设您已经知道基本的Java编程或简单应用。

# Java.io.BufferedInputStream类实例 - Java.io包

**Java.io.BufferedInputStream** 类添加功能到另一个输入流，缓冲输入以及支持mark和reset methods.Following是关于缓冲输入流的要点：

- 当创建缓冲输入，创建一个内部缓冲区数组。
- 如从该流的字节被读出或跳过，内部缓冲器被再从包含的输入流，许多字节一次必要的。

## 类的声明

以下是java.io.BufferedInputStream类的声明：

```
public class BufferedInputStream
   extends FilterInputStream
```

## 字段域

以下是java.io.BufferedInputStream类中的字段：

- protected byte[] buf -- 这是其中数据存储在内部缓冲器阵列。
- protected int count -- 这是该指数1大于在缓冲器中的最后一个有效字节的索引。
- protected int marklimit -- 这是预读之前，后续调用reset方法失败调用mark方法后允许的最大值。
- protected int markpos -- 这是pos区域在最后标记方法被调用时的值。
- protected int pos -- 这是在缓冲器中的当前位置。
- protected InputStream in -- 这是将进行过滤的输入流。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **BufferedInputStream(InputStream in)** 这将创建一个缓冲输入并保存其参数在输入流中，以备后用。 |
| 2 | **BufferedInputStream(InputStream in, int size)** 这将创建具有指定缓冲区大小的一个缓冲输入流，并保存其参数在输入流中，以备后用。 |

# 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | int available() 此方法从这个输入流中可通过一个方法的下一次调用阻塞该输入流返回可以读取（或跳过）的字节数的估计值。 |
| 2 | void close() 此方法关闭此输入流并释放与该流关联的所有系统资源。 |
| 3 | void mark(int readlimit) 此方法请参阅InputStream的mark方法的常规协定。 |
| 4 | boolean markSupported() 如果此输入流是否支持mark和reset方法的方法测试。 |
| 5 | int read() 此方法读取从输入流中的下一个数据字节。 |
| 6 | int read(byte[] b, int off, int len) 此方法读取该字节输入流中的字节到指定的字节数组，并从给定的偏移量。 |
| 7 | void reset() 此方法重新定位这个流，以当时的mark方法最后调用这个输入流中的位置。 |
| 8 | long skip(long n) 此方法跳过并丢弃n个字节从此输入流中的数据。 |

# 继承的方法

这个类继承自以下类方法：

- Java.io.FilterInputStream
- Java.io.Object

# Java.io.BufferedOutputStream类使用例子 - Java.io包

**Java.io.BufferedOutputStream** 类实现一个缓冲输出流。通过建立这样一个输出流，应用程序可以写字节到底层输出流，而不必然导致调用用底层的系统写入的每个字节。

## 类声明

以下是Java.io.BufferedOutputStream类的声明：

```
public class BufferedOutputStream
   extends FilterOutputStream
```

## 字段域

以下是Java.io.BufferedOutputStream类中的字段：

- protected byte[] buf -- 这是在数据被存储在内部缓冲器中。
- protected int count -- 这是在缓冲器中的有效字节数。
- protected OutputStream out -- 这是相关的输出流进行过滤。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **BufferedOutputStream(OutputStream out)** 这将创建一个新的缓冲输出流将数据写入到指定的基础输出流。 |
| 2 | **BufferedOutputStream(OutputStream out, int size)** 这将创建一个新的缓冲输出流的数据与指定的缓冲区大小写入指定的基础输出流。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void flush() 这个方法刷新此缓冲的输出流。 |
| 2 | void write(byte[] b, int off, int len) 这个方法从指定的字节数组开始在这个缓冲的输出流关闭写入len字节。 |
| 3 | void write(int b) 此方法写入指定的字节写入此缓冲的输出流。 |

# 方法继承

这个类继承自以下类方法：

- Java.io.FilterOutputStream
- Java.io.Object

# Java.io.BufferedReader类 - Java.io包

Java.io.BufferedReader 类从字符输入流中读取文本，缓冲各个字符，从而提供字符，数组和行的高效读取。以下是有关的BufferedReader要点：

- 缓冲区的大小可以被指定或默认的大小也可使用。
- Reader的每一个读取请求会导致相应的读取请求底层字符或字节流。

## 类的声明

以下是java.io.BufferedReader类的声明：

```
public class BufferedReader
   extends Reader
```

## 字段

以下是java.io.BufferedReader类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类的构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **BufferedReader(Reader in)** 这将创建一个使用默认大小输入缓冲区的缓冲字符输入流。 |
| 2 | **BufferedReader(Reader in, int sz)** 这将创建一个使用指定大小输入缓冲区的缓冲字符输入流。 |

## 类方法

| S.N. | 方法与说明 |
|---|---|
| 1 | void close() 此方法关闭该流并释放与之关联的所有系统资源。 |
| 2 | void mark(int readAheadLimit) 此方法标记流中的当前位置。 |
| 3 | boolean markSupported() 这个方法告诉此流是否支持mark()操作，这确实如此。 |
| 4 | int read() 此方法读取单个字符。 |
| 5 | int read(char[] cbuf, int off, int len) 此方法读取字符到一个数组中的一部分。 |
| 6 | String readLine() 此方法读取一行文本。 |
| 7 | boolean ready() 这个方法告诉此流是否已准备好被读取。 |
| 8 | void reset() 这个方法重置流。 |
| 9 | long skip(long n) 此方法跳过n个字符。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Reader
- Java.io.Object

# Java.io.BufferedWriter类 - Java.io包

**Java.io.BufferedWriter** 类将文本写入字符输出流，缓冲各个字符，从而提供单个字符，数组和字符串的高效写入。以下是有关的BufferedWriter要点：

- 缓冲区的大小可以被指定或默认的大小也可使用。
- Writer 发送其输出到底层字符或字节流。

## 类的声明

以下是Java.io.BufferedWriter类的声明：

```
public class BufferedWriter
   extends Writer
```

## 字段

以下是Java.io.BufferedWriter类中的字段：

- protected Object lock -- 这是用于同步针对此流操作的对象。

## 类的构造函数

| S.N. | 构造函数与说明 |
| --- | --- |
| 1 | **BufferedWriter(Writer out)** 这将创建一个使用默认大小输出缓冲区的缓冲字符输出流。 |
| 2 | **BufferedWriter(Writer out, int sz)** 这将创建一个使用给定大小的输出缓冲区的新缓冲字符输出流。 |

## 类方法

| S.N. | 方法与说明 |
| --- | --- |
| 1 | void close() 此方法关闭该流，但要先刷新它。 |
| 2 | void flush() 这个方法刷新流。 |
| 3 | void newLine() 此方法写入一个行分隔符。 |
| 4 | void write(char[] cbuf, int off, int len) 此方法写入字符数组的一部分。 |
| 5 | void write(int c) 此方法写入的单个字符。 |
| 6 | void write(String s, int off, int len) 此方法写入一个字符串的一部分。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Writer
- Java.io.Object

# Java.io.ByteArrayInputStream类 - Java.io包

**java.io.ByteArrayInputStream** 类包含包含可从流中读取的字节的内部缓冲器。内部计数器保持下一个字节的读法提供的轨道。以下是关于ByteArrayInputStream类重要的几点：

- 关闭ByteArrayInputStream类没有任何效果。
- 在这个类中的方法可以在流已关闭后，而被调用不会产生一个IOException。

## 类的声明

以下是java.io.ByteArrayInputStreamx类的声明：

```
public class ByteArrayInputStream
   extends InputStream
```

## 字段

以下是java.io.ByteArrayInputStream中类中的字段：

- protected byte[] buf -- 这是由流的创建者所提供的字节数组。
- protected int count -- 这是索引一个大于在输入流缓冲器中的最后一个有效字符。
- protected int mark -- 这是流中的当前标记的位置。
- protected int pos -- 这是从输入流缓存器读取的下一个字符索引。

## 类的构造函数

| S.N. | 构造函数与说明 |
|---|---|
| 1 | **ByteArrayInputStream(byte[] buf)** 这将创建一个ByteArrayInputStream类，以便它使用的buf为缓冲区数组。 |
| 2 | **ByteArrayInputStream(byte[] buf, int offset, int length)** 这将创建一个ByteArrayInputStream类使用的buf为缓冲区数组。 |

## 类方法

| S.N. | 方法与说明 |
| --- | --- |
| 1 | int available() 这个方法从当前输入流返回可以读取(或跳过)的剩余字节数。 |
| 2 | void close() 关闭ByteArrayInputStream类没有任何影响效果。 |
| 3 | void mark(int readAheadLimit) 这种方法在流中设置的当前标记的位置。 |
| 4 | boolean markSupported() 是否这个InputStream支持标记/重置此方法测试。 |
| 5 | int read() 此方法读取从这个输入流数据的下一个字节。 |
| 6 | int read(byte[] b, int off, int len) 此方法读取最多len个字节数据到从这个输入流中的字节数组。 |
| 7 | void reset() 这种方法缓冲区重置为标记位置。 |
| 8 | long skip(long n) 此方法跳过输入流中n个字节的输入。 |

## 继承的方法

这个类继承自以下类方法：

- java.io.InputStream
- java.io.Object

# Java.io.ByteArrayOutputStream类 - Java.io包

**Java.io.ByteArrayOutputStream** 类实现输出流中的数据被写入一个字节数组。作为数据写入到它的缓冲自动增长。以下是有关的ByteArrayOutputStream要点：

- 关闭一个字节数组输出流没有影响。
- 在这个类中的方法可以在流已关闭后被调用，不会产生一个IOException。

## 类的声明

以下是java.io.ByteArrayOutputStream声明的类：

```
public class ByteArrayOutputStream
   extends OutputStream
```

## 字段

以下是java.io.ByteArrayOutputStream中类中的字段：

- protected byte[] buf -- 这是在数据被存储在缓冲器中。
- protected int count -- 这是在缓冲器中的有效字节数。

## 类的构造函数

| S.N. | 构造函数与说明 |
|---|---|
| 1 | **ByteArrayOutputStream()** 这将创建一个新的字节数组输出流。 |
| 2 | **ByteArrayOutputStream(int size)** 这将创建一个新的字节数组输出流，具有缓冲容量指定的大小，以字节为单位。 |

## 类方法

| S.N. | 方法与说明 |
|---|---|
| 1 | void close() 关闭一个字节数组输出流没有影响。 |
| 2 | void reset() 此方法重置该字节数组输出流为零的计数字段，以便在输出流中的所有当前累计输出被丢弃。 |
| 3 | int size() 此方法返回缓冲区的当前大小。 |
| 4 | byte[] toByteArray() 此方法创建一个新分配的字节数组。 |
| 5 | String toString() 这种方法将缓冲区的内容转换为使用平台的默认字符集的字符串解码字节。 |
| 6 | String toString(String charsetName) 此方法通过使用指定charsetName解码字节将缓冲区的内容转换成一个字符串。 |
| 7 | void write(byte[] b, int off, int len) 这个方法从指定的字节数组开始在该字节数组输出流关闭写入len字节。 |
| 8 | void write(int b) 这种方法将指定字节写入该字节数组输出流。 |
| 9 | void writeTo(OutputStream out) 此方法写入该字节数组输出流的全部内容写入到指定的输出流参数，因为这与使用out.write(buf, 0, count)调用输出流的write方法。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.OutputStream
- Java.io.Object

# Java.io.CharArrayReader类 - Java.io包

**Java.io.CharArrayReader** 类实现，可以用来作为一个字符输入流的字符缓冲区。

## 类声明

以下是java.io.CharArrayReader类的声明：

```
public class CharArrayReader
   extends Reader
```

## 字段

以下是java.io.CharArrayReader中类中的字段：

- protected char[] buf -- 这是字符缓冲区。
- protected int count -- 这是此缓冲区的末尾的索引。
- protected int markedPos -- 这是标记在缓冲区中的位置。
- protected int pos -- 这是当前的缓冲区位置。
- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类的构造函数

| S.N. | 构造函数与说明 |
| --- | --- |
| 1 | **CharArrayReader(char[] buf)** 这将创建由字符指定一个数组CharArrayReader。 |
| 2 | **CharArrayReader(char[] buf, int offset, int length)** 这将创建由字符指定一个数组CharArrayReader。 |

## 类方法

| S.N. | 方法与说明 |
|---|---|
| 1 | void close() 此方法关闭该流并释放与之关联的所有系统资源。 |
| 2 | void mark(int readAheadLimit) 此方法标记流中的当前位置。 |
| 3 | boolean markSupported() 这个方法告诉此流是否支持mark()操作。 |
| 4 | int read() 此方法读取单个字符。 |
| 5 | int read(char[] b, int off, int len) 该方法读取字符到数组的一部分。 |
| 6 | boolean ready() 此方法通知此流是否已准备好被读取。 |
| 7 | void reset() 此方法重置流至最新的标记，或者如果它从来没有被标记的开始。 |
| 8 | long skip(long n) 此方法跳过n个字符。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Reader
- Java.io.Object

# Java.io.CharArrayWriter类 - Java.io包

**Java.io.CharArrayWriter** 类可以用来作为一个Writer的字符缓冲区。当数据被写入到流缓冲区会自动增长。

## 类声明

以下是java.io.CharArrayWriter类的声明：

```
public class CharArrayWriter
   extends Writer
```

## 字段

以下是**java.io.CharArrayWriter**类中的字段：

- protected char[] buf -- 这是被存储在缓冲器中的数据。
- protected int count -- 这是在缓冲区字符的数目。
- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类的构造函数

| S.N. | 构造函数与说明 |
|---|---|
| 1 | **CharArrayWriter()** 这将创建由字符指定数组一个CharArrayReader。 |
| 2 | **CharArrayWriter(int initialSize)** 这将创建一个新的CharArrayWriter使用指定的初始大小。 |

## 类方法

| S.N. | 方法与说明 |
|---|---|
| 1 | CharArrayWriter append(char c) 此方法将指定字符追加到这个writer。 |
| 2 | CharArrayWriter append(CharSequence csq) 此方法将指定的字符序列追加到这个writer。 |
| 3 | CharArrayWriter append(CharSequence csq, int start, int end) 此方法将指定的字符序列的子序列写入此writer。 |
| 4 | void close() 这种方法关闭流。 |
| 5 | void flush() 这个方法刷新流。 |
| 6 | void reset() 此方法重设缓冲区，这样就可以再次使用它而无需丢弃已分配的缓冲区。 |
| 7 | int size() 此方法返回缓冲区的当前大小。 |
| 8 | char[] toCharArray() 此方法返回输入数据的副本。 |
| 9 | String toString() 这种方法的输入数据转换为字符串。 |
| 10 | void write(char[] c, int off, int len) 此方法写入字符到缓冲区。 |
| 11 | void write(int c) 这种方法将一个字符写入到缓冲区。 |
| 12 | void write(String str, int off, int len) 此方法写入的字符串的一部分到缓冲区。 |
| 13 | void writeTo(Writer out) 此方法写入的缓冲区的内容到另一个字符流。 |

## 继承的方法

这个类从以下类继承的方法：

- Java.io.Writer
- Java.io.Object

# Java.io.Console类 - Java.io包

Java.io.Console 类提供了方法来访问基于字符的控制台设备，与当前Java虚拟机相关联。

## 类声明

以下是Java.io.Console类的声明：

```
public final class Console
  extends Object
    implements Flushable
```

## 类方法

| S.N. | 方法与说明 |
|---|---|
| 1 | void flush() 此方法刷新控制台，并强制所有缓冲的输出立即写入。 |
| 2 | Console format(String fmt, Object... args) 此方法写入一个格式化字符串使用指定格式字符串和参数这个控制台的输出流。 |
| 3 | Console printf(String format, Object... args) 此方法是用来写一个格式化字符串使用指定格式字符串和参数这个控制台的输出流。 |
| 4 | Reader reader() 此方法检索与此控制台关联的唯一Reader对象。 |
| 5 | String readLine() 此方法从控制台读取单行文本。 |
| 6 | String readLine(String fmt, Object... args) 此方法提供了一个格式化提示，然后从控制台读取单行文本。 |
| 7 | char[] readPassword() 此方法从控制台读取密码或口令，禁用回显。 |
| 8 | char[] readPassword(String fmt, Object... args) 此方法提供了一个格式化提示，然后读取密码或口令从控制台禁用回显。 |
| 9 | PrintWriter writer() 此方法检索与此控制台关联的唯一PrintWriter对象。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.DataInputStream类 - Java.io包

Java.io.DataInputStream 类允许应用程序读取在与机器无关方式从底层输入流基本Java数据类型。以下是有关数据输入流的要点：

- 应用程序使用数据输出流写入，以后可以通过一个数据输入流中读取数据。
- 数据输入流并不一定是安全的多线程访问。线程安全是可选的，在这个类中的方法用户的责任。

## 类声明

以下是java.io.DataInputStream类的声明：

```
public class DataInputStream
   extends FilterInputStream
      implements DataInput
```

## 字段

以下是java.io.DataInputStream类中的字段：

- **protected InputStream in** -- 这是将输入流进行过滤。

## 类的构造函数

| S.N. | 构造函数与说明 |
|---|---|
| 1 | **DataInputStream(InputStream in)** 这将创建一个DataInputStream使用指定的底层InputStream。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int read(byte[] b) 此方法从包含的输入流中读取字节数部分，并将它们存储到缓冲区数组b |
| 2 | int read(byte[] b, int off, int len) 此方法读取最多len个从包含的输入流的数据字节为字节数组。 |
| 3 | boolean readBoolean() 此方法读取一个输入字节，如果该字节不为零返回true，如果该字节是零则返回false。 |
| 4 | byte readByte() 此方法读取并返回一个输入字节。 |
| 5 | char readChar() 此方法读取两个输入字节并返回一个char值。 |
| 6 | double readDouble() 此方法读取八个输入字节并返回一个double值。 |
| 7 | float readFloat() 此方法读取四个输入字节并返回一个float值。 |
| 8 | void readFully(byte[] b) 此方法读取从输入流的一些字节，并将它们存储到缓冲区数组b中。 |
| 9 | void readFully(byte[] b, int off, int len) 此方法从输入流读取len个字节。 |
| 10 | int readInt() 此方法读取四个输入字节并返回一个int值。 |
| 11 | long readLong() 此方法读取八个输入字节并返回一个长整型值。 |
| 12 | short readShort() 此方法读取两个输入字节并返回一个short值。 |
| 13 | int readUnsignedByte() 此方法读取一个输入字节，零扩展到int类型，并返回结果，所以结果的范围是从0到255。 |
| 14 | int readUnsignedShort() 此方法读取两个输入字节，并通过返回在范围0-65535的int值。 |
| 15 | String readUTF() 此方法读取在已使用UTF-8修改版格式编码的字符串。 |
| 16 | static String readUTF(DataInput in) 此方法读取来自编码经修订的UTF-8格式的Unicode字符串的表示数据流;这串字符接着返回一个字符串。 |
| 17 | int skipBytes(int n) 此方法使得试图从输入流中的数据跳过n个字节，丢弃跳过的字节。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.FilterInputStream
- Java.io.Object

# Java.io.DataOutputStream类 - Java.io包

**Java.io.DataOutputStream** 类允许应用程序写入基本Java数据类型在一个可移植的方式输出流。然后，应用程序可以使用数据输入流中读取的数据回入。

## 类声明

以下是java.io.DataOutputStream类的声明：

```
public class DataOutputStream
   extends FilterOutputStream
      implements DataOutput
```

## 字段

以下是java.io.DataOutputStream类的字段：

- protected int written -- 这是写入到数据输出流到目前为止的字节数。
- protected OutputStream out -- 这是相关的输出流进行过滤。

## 类的构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **DataOutputStream(OutputStream out)** 这将创建一个新的数据输出流将数据写入到指定的基础输出流。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void flush() 此方法刷新此数据输出流。 |
| 2 | int size() 此方法返回计数器的当前值写入，写入该数据输出流到目前为止的字节数。 |
| 3 | void write(byte[] b, int off, int len) 此方法从指定的字节数组开始到底层输出流关闭写入len字节。 |
| 4 | void write(int b) 此方法写入指定的字节(低8位参数b)底层输出流。 |
| 5 | void writeBoolean(boolean v) 此方法写入一个布尔到底层输出流为1个字节的值。 |
| 6 | void writeByte(int v) 此方法写入了一个字节到基础输出流中1个字节的值。 |
| 7 | void writeBytes(String s) 此方法写出的字符串到底层输出流为字节序列。 |
| 8 | void writeChar(int v) 此方法写入一个char到底层输出流作为一个2-byte值，高字节在前。 |
| 9 | void writeChars(String s) 此方法将一个字符串写入基础输出流作为一个字符序列。 |
| 10 | void writeDouble(double v) 此方法将float参数转换为使用Float类的floatToIntBits方法，写入int值到底层输出流作为一个4字节的数量，高字节在前。 |
| 11 | void writeFloat(float v) 此方法将float参数转换为使用Float类的floatToIntBits方法，写入int值到底层输出流作为一个4字节的数量，高字节在前。 |
| 12 | void writeInt(int v) 此方法写入一个int到底层输出流为4个字节，高字节在前。 |
| 13 | void writeLong(long v) 此方法写入了long的基础输出流中的8个字节，高字节在前。 |
| 14 | void writeShort(int v) 此方法写入了短到底层输出流为两个字节，高字节在前。 |
| 15 | void writeUTF(String str) 此方法将一个字符串写入使用经修订的UTF-8编码以与机器无关的方式的基础输出流。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.FilterOutputStream

- Java.io.Object
- Java.io.DataOutput

# Java.io.File类 - Java.io包

**java.io.File**类是文件和目录路径名的抽象表示。以下是有关文件的要点：

- 实例可以或可以不表示实际的文件系统对象，如文件或目录。如果是这样表示这样一个对象，然后该对象位于一个分区。分区是存储为文件系统的操作系统的特定部分。
- 文件系统可以实现限制某些操作的实际文件系统对象，如读，写，和执行上。这些限制统称为访问权限。
- File类的实例是不可变的;也就是说，一旦创建，由一个File对象表示的抽象路径名是不会改变的。

## 类的声明

以下是声明java.io.File类：

```
public class File
  extends Object
    implements Serializable, Comparable<File>
```

## 字段

以下是java.io.File类的字段：

- static String pathSeparator -- 这是系统相关的路径分隔符，表示为一个字符串以方便使用。
- static char pathSeparatorChar -- 这是依赖于系统的路径分隔符。
- static String separator -- 这是与系统有关的默认名称分隔符，表示为一个字符串以方便使用。
- static char separatorChar -- 这是与系统有关的默认名称分隔符。

## 类的构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **File(File parent, String child)** 此方法创建从父抽象路径名和child路径名字符串的新File实例。 |
| 2 | **File(String pathname)** 该方法通过将给定路径名字符串转换为抽象路径名来创建一个新File实例。 |
| 3 | **File(String parent, String child)** 此方法创建从父路径名字符串和child路径名字符串的新File实例。 |
| 4 | **File(URI uri)** URI转换成抽象路径名：此方法通过给定的文件将创建一个新的File实例。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean canExecute() 此方法测试应用程序是否可以执行表示此抽象路径名的文件。 |
| 2 | boolean canRead() 这种方法测试应用程序是否可以读取表示此抽象路径名的文件。 |
| 3 | boolean canWrite() 此方法测试应用程序是否可以修改表示此抽象路径名的文件。 |
| 4 | int compareTo(File pathname) 这种方法比较两个抽象路径名的字典顺序。 |
| 5 | boolean createNewFile() 此方法自动创建此抽象路径名命名的，当且仅当具有此名称的文件尚不存在一个新的空文件。 |
| 6 | static File createTempFile(String prefix, String suffix) 此方法创建的默认临时文件目录的空文件，使用给定前缀和后缀生成其名称。 |
| 7 | static File createTempFile(String prefix, String suffix, File directory) 此方法会在指定的目录中一个新的空文件，使用给定前缀和后缀字符串生成其名称。 |
| 8 | boolean delete() 此方法删除表示此抽象路径名的文件或目录。 |
| 9 | void deleteOnExit() 此方法要求将表示此抽象路径名的文件或目录在虚拟机终止时被删除。 |
| 10 | boolean equals(Object obj) 此方法测试此抽象路径名与给定对象是否相等。 |
| 11 | boolean exists() 此方法测试表示此抽象路径名的文件或目录是否存在。 |
| 12 | File getAbsoluteFile() 此方法返回此抽象路径名的绝对形式。 |
| | |

| | |
|---|---|
| 13 | String getAbsolutePath() 此方法返回此抽象路径名的绝对路径名字符串。 |
| 14 | File getCanonicalFile() 此方法返回此抽象路径名的规范形式。 |
| 15 | String getCanonicalPath() 此方法返回此抽象路径名的规范路径名字符串。 |
| 16 | long getFreeSpace() 此方法返回此抽象路径名的分区中的未分配的字节数。 |
| 17 | String getName() 此方法返回表示此抽象路径名的文件或目录的名称。 |
| 18 | String getParent() 此方法返回此抽象路径名的父路径名的字符串，或者null，如果此路径名没有指定父目录。 |
| 19 | File getParentFile() 此方法返回此抽象路径名的父抽象路径名，或null，如果此路径名没有指定父目录。 |
| 20 | String getPath() 此方法此抽象路径名转换为一个路径名字符串。 |
| 21 | long getTotalSpace() 此方法返回此抽象路径名的分区的大小。 |
| 22 | long getUsableSpace() 此方法返回可用字节数这个虚拟机上命名此抽象路径名的分区。 |
| 23 | int hashCode() 此方法用于计算此抽象路径名的哈希码。 |
| 24 | boolean isAbsolute() 此方法测试此抽象路径名是否是绝对的。 |
| 25 | boolean isDirectory() 此方法测试表示此抽象路径名的文件是否是一个目录。 |
| 26 | boolean isFile() 此方法测试表示此抽象路径名的文件是否是一个正常的文件。 |
| 27 | boolean isHidden() 此方法测试此抽象路径名的文件是否是一个隐藏文件。 |
| 28 | long lastModified() 此方法返回的时候，表示此抽象路径名的文件的最后修改 |
| 29 | long length() 此方法返回表示此抽象路径名的文件的长度。 |
| 30 | String[] list() 此方法返回的字符串命名表示此抽象路径名的目录中的文件和目录的数组。 |
| 31 | String[] list(FilenameFilter filter) 此方法返回的字符串命名的目录表示此抽象路径名满足指定过滤器的文件和目录的数组。 |
| 32 | File[] listFiles() 此方法返回抽象路径名表示在表示此抽象路径名的目录中的文件的数组。 |
| 33 | File[] listFiles(FileFilter filter) 此方法返回抽象路径名表示的目录表示此抽象路径名满足指定过滤器的文件和目录的数组。 |

| | |
|---|---|
| 34 | File[] listFiles(FilenameFilter filter) 此方法返回抽象路径名表示的目录表示此抽象路径名满足指定过滤器的文件和目录的数组。 |
| 35 | static File[] listRoots() 此方法列出可用的文件系统的根。 |
| 36 | boolean mkdir() 此方法创建此抽象路径名的目录。 |
| 37 | boolean mkdirs() 此方法创建此抽象路径名的目录，包括任何必需但不存在的父目录 |
| 38 | boolean renameTo(File dest) 这种方法将重命名表示此抽象路径名的文件。 |
| 39 | boolean setExecutable(boolean executable) 这是一个方便的方法来设置所有者对于此抽象路径名执行权限。 |
| 40 | boolean setExecutable(boolean executable, boolean ownerOnly) 此方法设置所有者或每个人的执行权限，此抽象路径名。 |
| 41 | boolean setLastModified(long time) 此方法设置此抽象路径名的文件或目录的最后修改时间。 |
| 42 | boolean setReadable(boolean readable) 这是一个方便的方法来设置此抽象路径名的所有者的读取权限。 |
| 43 | boolean setReadable(boolean readable, boolean ownerOnly) 此方法设置所有者或在此抽象路径名大家的读取权限。 |
| 44 | boolean setReadOnly() 此方法标志着此抽象路径名命名的，这样只允许读操作的文件或目录。 |
| 45 | boolean setWritable(boolean writable) 这是一个方便的方法来设置此抽象路径名的所有者的写权限。 |
| 46 | boolean setWritable(boolean writable, boolean ownerOnly) 此方法设置此抽象路径名的所有者或每个人的写权限。 |
| 47 | String toString() 此方法返回此抽象路径名的路径名字符串。 |
| 48 | URI toURI() 这种方法构造一个文件：URI表示此抽象路径名。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.FileDescriptor类 - Java.io包

**Java.io.FileDescriptor** 类实例作为一个不透明的句柄底层机器特有的结构表示一个打开的文件，打开的套接字或其他来源或字节的接收器。以下是关于FileDescriptor要点：

- 主要实际使用的文件描述符是创建一个FileInputStream或FileOutputStream来遏制它。
- 应用程序不应创建自己的文件描述符。

## 类的声明

以下是java.io.FileDescriptor类的声明：

```
public final class FileDescriptor
   extends Object
```

## 字段

以下是java.io.FileDescriptor类中的字段：

- static FileDescriptor err -- 这是句柄的标准错误流。
- static FileDescriptor in -- 这是句柄的标准输入流。
- static FileDescriptor out -- 这是句柄到标准输出流。

## 类的构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **FileDescriptor()** 这种方法构造一个(无效)FileDescriptor对象。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | void sync() 此方法强制所有系统缓冲区与基础设备同步。 |
| 1 | boolean valid() 如果此方法测试文件描述符对象是否有效？ |

# 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.FileInputStream类 - Java.io包

Java.io.FileInputStream 类从文件系统中的一个文件中获取输入字节。那些文件依赖于主机环境。以下是关于文件输入流的要点：

- 这个类是指对原始字节诸如图像数据读出流。
- 对于字符读取流，使用FileReader。

## 类的声明

以下是java.io.FileInputStream类的声明：

```
public class FileInputStream
   extends InputStream
```

## 类构造方法

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **FileInputStream(File file)** 这通过打开一个到实际文件，命名在文件系统中的File对象文件的文件的连接创建一个FileInputStream。 |
| 2 | **FileInputStream(FileDescriptor fdObj)** 这通过使用文件描述符fdObj，它代表在文件系统中某个实际文件的现有连接创建一个FileInputStream。 |
| 3 | **FileInputStream(String name)** 这将创建一个FileInputStream通过打开一个到实际文件的连接，通过路径名名在文件系统命名的文件。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 此方法从输入流中通过一个方法的下一次调用阻塞该输入流返回可以读取（或跳过）的剩余字节数的估计值。 |
| 2 | void close() 此方法关闭此文件输入流并释放与该流关联的所有系统资源。 |
| 3 | protected void finalize() 此方法可确保此文件输入流的close方法被调用当它没有更多的参考引用。 |
| 4 | FileChannel getChannel() 此方法返回与此文件输入流关联的唯一文件通道对象。 |
| 5 | FileDescriptor getFD() 此方法返回FileDescriptor对象，表示连接到正在使用此文件输入流文件系统的实际文件。 |
| 6 | int read() 此方法读取当前输入流中一个字节的数据。 |
| 7 | int read(byte[] b) 此方法从这个输入流中数据读取可达b.length个字节到字节数组。 |
| 8 | int read(byte[] b, int off, int len) 此方法读取最多len个从这个输入流中数据的字节到字节数组。 |
| 9 | long skip(long n) 此方法跳过并丢弃n个字节从输入流中的数据。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.InputStream
- Java.io.Object

# Java.io.FileOutputStream类 - Java.io包

**Java.io.FileOutputStream** 类是用于将数据写入一个文件或FileDescriptor的输出流。以下是关于文件输出流的要点：

- 这个类是指用于记录的原始字节，例如图像数据流。
- 写字符流，可以使用文件字符 FileWriter

## 类声明

以下是java.io.FileOutputStream类的声明：

```
public class FileOutputStream
   extends OutputStream
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **FileOutputStream(File file)** 这将创建一个文件输出流写入到由指定的File对象表示文件。 |
| 2 | **FileOutputStream(File file, boolean append)** 这将创建一个文件输出流写入到由指定的File对象表示的文件。 |
| 3 | **FileOutputStream(FileDescriptor fdObj)** 这将创建一个输出文件流写入到指定的文件描述符，它代表了文件系统中的某个实际文件的现有连接。 |
| 4 | **FileOutputStream(String name)** 这将创建一个输出文件流写入到具有指定名称的文件。 |
| 5 | **FileOutputStream(String name, boolean append)** 这将创建一个输出文件流写入到具有指定名称的文件。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭此文件输出流并释放与此流有关的所有系统资源。 |
| 2 | protected void finalize() 这种方法清除连接到文件，并确保此文件输出流的close方法被调用时，此流不再有引用。 |
| 3 | FileChannel getChannel() 此方法返回与此文件输出流关联的唯一文件通道对象。 |
| 4 | FileDescriptor getFD() 此方法返回与此流有关的文件描述符。 |
| 5 | void write(byte[] b) 此方法写入b.length个字节从指定的字节数组到该文件输出流。 |
| 6 | void write(byte[] b, int off, int len) 此方法从指定的字节数组开始到该文件输出流关闭写入len字节。 |
| 7 | void write(int b) 此方法写入指定的字节写入此文件输出流。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.OutputStream
- Java.io.Object

# Java.io.FilePermission类 - Java.io包

**Java.io.FilePermission** 类表示访问一个文件或目录。它包括一个路径名和一套有效的路径名操作。以下是有关的FilePermission要点：

- 将予授出的动作传递给构造函数中包含一个或多个逗号分隔的关键字列表的字符串。可能的关键字是"read", "write", "execute", 和"delete"。
- 代码总是可以从同一目录它在(或该目录的子目录)读取文件;它并不需要明确许可这样做。

## 类的声明

以下是Java.io.FilePermission类的声明：

```
public final class FilePermission
   extends Permission
      implements Serializable
```

## 类的构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **FilePermission(String path, String actions)** 将创建具有指定操作新的FilePermission对象。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean equals(Object obj) 此方法检查两个FilePermission对象是否相等。 |
| 2 | String getActions() 此方法返回动作的“规范化字符串表示形式”。 |
| 3 | int hashCode() 此方法返回此对象的哈希码值。 |
| 4 | boolean implies(Permission p) 此方法检查，FilePermission对象是否“暗含”指定的权限。 |
| 5 | PermissionCollection newPermissionCollection() 此方法返回一个新的对象PermissionCollection存储的FilePermission对象。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Permission
- Java.io.Object

# Java.io.FileReader类 - Java.io包

**Java.io.FileReader** 类是一个方便的类用于读取字符文件。以下是有关的FileReader要点：

- 这个类的构造方法假定默认字符编码和默认字节缓冲区大小都是适当的。
- FileReader是为字符读取流。对于原始字节流读取，使用文件输入流-FileInputStream。

## 类的声明

以下是Java.io.FileReader类的声明：

```
public class FileReader
   extends InputStreamReader
```

## 字段

以下是Java.io.FileReader类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类的构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **FileReader(File file)** 创建一个新的FileReader，从给定的文件读取。 |
| 2 | **FileReader(FileDescriptor file)** 创建一个新的FileReader，从给出的FileDescriptor读取。 |
| 3 | **FileReader(String fileName)** 创建一个新的FileReader，因为要读取的文件的名称。 |

## 类方法

这个类继承自以下类方法：

- Java.io.InputStreamReader
- java.uti.Reader

- Java.io.Object

# Java.io.FileWriter类 - Java.io包

**Java.io.FileWriter** 类是一个方便的类写入字符文件。以下是关于FileWriter中重要的几点：

- 这个类的构造方法假定默认字符编码和默认字节缓冲区大小都是可以接受的。
- 文件字符写的意思是写入字符流。对于原始字节写入流，使用文件输出流。

## 类声明

以下是Java.io.FileWriter类的声明：

```
public class FileWriter
   extends OutputStreamReader
```

## 字段

以下是Java.io.FileWriter类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **FileWriter(File file)** 此构造一个文件字符写对象给出一个File对象。 |
| 2 | **FileWriter(File file, boolean append)** 此构造一个文件字符写对象给出一个File对象。 |
| 3 | **FileWriter(FileDescriptor fd)** 此构造与文件描述符相关联的文件字符写的对象。 |
| 4 | **FileWriter(String fileName)** 此构造一个文件字符写对象给出文件名。 |
| 5 | **FileWriter(String fileName, boolean append)** 此构造函数一个文件字符写对象给出一个布尔值，指示是否附加写入数据的文件名。 |

## 类方法

这个类继承自以下类方法：

- Java.io.OutputStreamWriter
- java.uti.Writer
- Java.io.Object

# Java.io.FilterInputStream类 - Java.io包

**Java.io.FilterInputStream** 类包含其他一些输入流，它用作其基本数据源，它可以直接传输数据或提供一些额外的功能。以下是关于FilterInputStream中的要点：

- 类本身只是简单地重写InputStream与所有请求传递给所包含的输入流版本的所有方法
- 这个类的子类可进一步重写其中的一些方法，还可能提供额外的方法和字段。

## 类声明

以下是java.io.FilterInputStream类的声明：

```
public class FilterInputStream
   extends InputStream
```

## 字段

以下是java.io.FilterInputStream类中的字段：

- protected InputStream in -- 此是将输入流进行过滤。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected FilterInputStream(InputStream in)** 在到字段这个分配参数创建一个FilterInputStream。并记住它供以后使用。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 此方法从这个输入流中可通过一个方法的调用者阻止这个输入流返回可以读取(或跳过)的字节数的估计值。 |
| 2 | void close() 此方法关闭此输入流并释放与该流关联的所有系统资源。 |
| 3 | void mark(int readlimit) 此方法标志着在此输入流的当前位置。 |
| 4 | boolean markSupported() 如果此输入流是否支持mark和reset方法此方法测试。 |
| 5 | int read() 此方法读取从这个输入流数据的下一个字节。 |
| 6 | int read(byte[] b) 此方法从这个输入流中数据的读取byte.length字节到字节数组。 |
| 7 | int read(byte[] b, int off, int len) 此方法从这个输入流中数据读取最多len个字节到字节数组。 |
| 8 | void reset() 此方法重新定位此流，以当时的mark方法最后调用这个输入流中的位置。 |
| 9 | long skip(long n) 此方法从此输入流中的数据跳过并丢弃n个字节。 |

# 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.FilterOutputStream类 - Java.io包

**Java.io.FilterOutputStream** 类是所有类过滤器的输出流的超类。以下是关于FilterOutputStream的要点：

- 类本身只是简单地重写OutputStream的所有方法与所有请求传递给所包含的输出流的版本。
- 这个类的子类可进一步重写其中的一些方法，还可能提供额外的方法和字段。

## 类声明

以下是java.io.FilterOutputStream类的声明：

```
public class FilterOutputStream
   extends OutputStream
```

## 字段

以下是java.io.FilterOutputStream类中的字段：

- protected OutputStream out -- 这是输出流进行过滤。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **FilterOutputStream(OutputStream out)** 这将创建基于指定的基础输出流之上的输出流过滤器。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | void close() 此方法关闭此输出流并释放与该流关联的所有系统资源。 |
| 2 | void flush() 此方法刷新此输出流并强制将所有缓冲的输出字节被写出到流中。 |
| 3 | void write(byte[] b) 此方法写入b.length个字节到输出流。 |
| 4 | void write(byte[] b, int off, int len) 此方法从指定的字节数组开始到当前输出流关闭写入len个字节。 |
| 5 | void write(int b) 此方法将指定字节写入此输出流。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.FilterWriter类 - Java.io包

**Java.io.FilterWriter** 类是用于写入已过滤的字符流。以下是关于FilterWriter要点：

- 类本身提供了所有请求传递给所包含的流的默认方法。
- FilterWriter子类应重写其中的一些方法，还可能提供额外的方法和字段。

## 类声明

以下是Java.io.FilterWriter类的声明：

```
public abstract class FilterWriter
  extends Writer
```

## 字段域

以下是Java.io.FilterWriter类中的字段：

- protected Writer in -- 这是字符输出流。
- protected Object lock -- 这是用于同步针对此流的操作对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected FilterWriter(Writer in)** 这将创建一个新的过滤writer。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 该方法刷关闭流。 |
| 2 | void flush() 该方法刷新流。 |
| 3 | void write(char[] cbuf, int off, int len) 此方法写入字符数组的一部分。 |
| 4 | void write(int c) 此方法写入的单个字符。 |
| 5 | void write(String str, int off, int len) 该方法写入一个字符串的一部分。 |

# 方法继承

这个类从以下类继承的方法：

- Java.io.Writer
- Java.io.Object

# Java.io.InputStream类 - Java.io包

**Java.io.InputStream** 类是表示字节输入流的所有类的超类。这需要定义InputStream的子类的应用程序必须始终提供返回输入的下一个字节的方法。

## 类声明

以下是java.io.InputStream类的声明：

```
public abstract class InputStream
   extends Object
      implements Closeable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **InputStream()** 单个构造函数 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 此方法从这个输入流中可通过一个方法的下一次调用阻塞该输入流返回可以读取（或跳过）的字节数的估计值。 |
| 2 | void close() 此方法关闭此输入流并释放与该流关联的所有系统资源。 |
| 3 | void mark(int readlimit) 该方法标志在此输入流的当前位置。 |
| 4 | boolean markSupported() 如果此输入流是否支持mark和reset方法此方法测试。 |
| 5 | abstract int read() 此方法读取从输入流中的下一个数据字节。 |
| 6 | int read(byte[] b) 此方法读取一定数量的字节从输入流并将它们存储到缓冲区数组b中。 |
| 7 | int read(byte[] b, int off, int len) 此方法从输入流中读取多达len个数据的字节到字节数组。 |
| 8 | void reset() 此方法重新定位此流，当mark方法最后调用这个输入流中的位置。 |
| 9 | long skip(long n) 此方法从此输入流中跳过并丢弃n个字节的数据。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.InputStreamReader类 - Java.io包

**Java.io.InputStreamReader** 类是一座桥从字节流与字符流。它读取字节并将其解码为使用指定的字符集的字符。

## 类声明

以下是java.io.InputStreamReader类的声明：

```
public class InputStreamReader
   extends Reader
```

## 字段域

以下是java.io.InputStreamReader类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **InputStreamReader(InputStream in)** 这将创建一个使用默认字符集的输入流。 |
| 2 | **InputStreamReader(InputStream in, Charset cs)** 这将创建一个使用给定字符集的输入流。 |
| 3 | **InputStreamReader(InputStream in, CharsetDecoder dec)** 这将创建一个使用给定的charset解码器的输入流。 |
| 4 | **InputStreamReader(InputStream in, String charsetName)** 这将创建一个使用指定的字符集的输入流。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭该流并释放与之关联的所有系统资源。 |
| 2 | String getEncoding() 此方法返回正在使用此流的字符编码的名称。 |
| 3 | int read() 此方法读取单个字符。 |
| 4 | int read(char[] cbuf, int offset, int length) 此方法读取字符到一个数组中的一部分。 |
| 5 | boolean ready() 此方法通知此流是否已准备好被读取。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Reader
- Java.io.Object

# Java.io.LineNumberInputStream类 - Java.io包

**Java.io.LineNumberInputStream** 类是一个输入流过滤器，可提供跟踪当前的行号的增加的功能。 行是用一个回车符(' r')，换行符(' n')，或者一个回车符结束的字节序列后面紧跟一个换行符。

## 类声明

以下是java.io.LineNumberInputStream类的声明：

```
public class LineNumberInputStream
   extends Reader
```

## 字段域

以下是java.io.LineNumberInputStream类中的字段：

- protected InputStream in -- 这是将进行过滤的输入流。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **LineNumberInputStream(InputStream in)** 此构造一个读取指定的输入流的输入的新行号输入流。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 此方法返回可以从此输入流中可无阻塞读取的字节数。 |
| 2 | int getLineNumber() 此方法返回当前行号。 |
| 3 | void mark(int readlimit) 该方法标记在此输入流的当前位置。 |
| 4 | int read() 此方法读取从这个输入流数据的下一个字节。 |
| 5 | int read(byte[] b, int off, int len) 此方法从这个输入流中读取多达len个字节数据到字节数组。 |
| 6 | void reset() 这个方法重新定位此流，以当时的mark方法最后调用这个输入流中的位置。 |
| 7 | void setLineNumber(int lineNumber) 此方法设置行号以指定的参数。 |
| 8 | long skip(long n) 这种方法从此输入流中跳过并丢弃n个字节的数据。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.FilterInputStream
- Java.io.Object

# Java.io.LineNumberReader类 - Java.io包

**Java.io.LineNumberReader**类是跟踪行号的缓冲字符输入流。 一行被认为是由一个换行符(' '), 回车符(' ')或回车符中的任何一个被终止紧跟一个换行符。

## 类声明

以下是java.io.LineNumberReader类的声明：

```
public class LineNumberReader
   extends BufferedReader
```

## 字段

以下是java.io.LineNumberReader类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **LineNumberReader(Reader in)** 这将创建一个新的行号读取器，使用默认输入缓冲区的大小。 |
| 2 | **LineNumberReader(Reader in, int sz)** 这将创建一个新的行号读取器，将字符读入给定大小的缓冲区。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int getLineNumber() 此方法获取当前行号。 |
| 2 | void mark(int readAheadLimit) 此方法标记流中的当前位置。 |
| 3 | int read() 此方法读取单个字符。 |
| 4 | int read(char[] cbuf, int off, int len) 此方法读字符到一个数组的某一部分。 |
| 5 | String readLine() 此方法读取一行文本。 |
| 6 | void reset() 此方法重置流的最近标记。 |
| 7 | void setLineNumber(int lineNumber) 此方法设置的当前行数。 |
| 8 | long skip(long n) 此方法跳过n个字符。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.BufferedReader
- Java.io.Reader
- Java.io.Object

# Java.io.ObjectInputStream类 - Java.io包

**Java.io.ObjectInputStream**类反序列化的原始数据和以前写的对象使用一个ObjectOutputStream。以下是关于缓冲输入流的要点：

- 它是用来恢复先前序列化的对象。它确保所有对象的图形中从流中创建的类型匹配存在于Java虚拟机的类。
- 类加载使用标准的机制，作为必需的。

## 类声明

以下是java.io.ObjectInputStream类的声明：

```
public class ObjectInputStream
  extends InputStream
    implements ObjectInput, ObjectStreamConstants
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | protected ObjectInputStream() 这个子类，是完全重新实现ObjectInputStream到不必分配仅由ObjectInputStream实现私有数据提供了一种方法。 |
| 2 | ObjectInputStream(InputStream in) 这将创建一个ObjectInputStream从指定的InputStream 。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 此方法返回可以不受阻塞地读取的字节数。 |
| 2 | void close() 此方法关闭输入流。 |
| 3 | void defaultReadObject() 此方法读取当前类的从这个流中的非静态和非瞬态字段。 |
| 4 | protected boolean enableResolveObject(boolean enable) 此方法使该流，以允许要被替换从流中读取的对象。 |
| 5 | int read() 此方法读取一个字节的数据。 |

| | |
|---|---|
| 6 | int read(byte[] buf, int off, int len) 此方法读取到一个字节数组。 |
| 7 | boolean readBoolean() 此方法读取一个布尔值。 |
| 8 | byte readByte() 此方法读取一个8位字节。 |
| 9 | char readChar() 此方法返回16位字符。 |
| 10 | protected ObjectStreamClass readClassDescriptor() 这种方法读取序列化流的类描述符。 |
| 11 | double readDouble() 此方法读取一个64位双。 |
| 12 | ObjectInputStream.GetField readFields() 此方法从流中读取持久字段并使其可通过名称。 |
| 13 | float readFloat() 此方法读取一个32位浮点。 |
| 14 | void readFully(byte[] buf) 此方法读取字节，阻塞直到所有字节被读取。 |
| 15 | void readFully(byte[] buf, int off, int len) 此方法读取字节，阻塞直到所有字节被读取。 |
| 16 | int readInt() 此方法读取一个32位的整数。 |
| 17 | long readLong() 此方法读取一个64位长。 |
| 18 | Object readObject() 此方法从ObjectInputStream中读取对象。 |
| 19 | protected Object readObjectOverride() 这种方法被称为由ObjectOutputStream受信任子类使用受保护的无参数构造函数构造对象输出流。 |
| 20 | short readShort() 此方法读取一个16位的短类型。 |
| 21 | protected void readStreamHeader() 提供此方法允许子类读取并验证自己的数据流头。 |
| 22 | Object readUnshared() 此方法从ObjectInputStream中读取“非共享”对象。 |
| 23 | int readUnsignedByte() 此方法读取一个无符号8位字节。 |
| 24 | int readUnsignedShort() 此方法读取一个无符号的16位短类型。 |
| 25 | String readUTF() This method reads a String in modified UTF-8 format. |
| 26 | void registerValidation(ObjectInputValidation obj, int prio) 此方法注册对象的图形被返回之前进行验证。 |
| 27 | protected Class<?> resolveClass(ObjectStreamClass desc) 此方法加载本地类相当于指定的流类的描述。 |
| 28 | protected Object resolveObject(Object obj) 此方法允许ObjectInputStream受信任子类反序列化过程中一个对象替代另一个。 |

| | |
|---|---|
| 29 | protected Class<?> resolveProxyClass(String[] interfaces) 此方法返回一个实现在代理类的描述符指定接口的代理类;子类可以实现此方法，从随描述符和动态代理类流中读取自定义数据，允许它们使用的替换加载机制的接口和代理类。 |
| 30 | int skipBytes(int len) 此方法跳过len个字节。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.InputStream
- Java.io.Object
- Java.io.ObjectInput

# Java.io.ObjectInputStream.GetField类 - Java.io包

**Java.io.ObjectInputStream.GetField** 类提供对从输入流中读取持久字段。

## 类声明

以下是Java.io.ObjectInputStream.GetField类的声明：

```
public abstract static class ObjectInputStream.GetField
  extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **ObjectInputStream.GetField()** Single Constructor |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | abstract boolean defaulted(String name) 此方法返回true，如果指定的字段是默认的，也没有这个值在流中。 |
| 2 | abstract boolean get(String name, boolean val) 此方法从持久字段获取指定的布尔字段的值。 |
| 3 | abstract byte get(String name, byte val) 此方法从持久字段获取指定字节字段的值。 |
| 4 | abstract char get(String name, char val) 此方法从持久字段获取指定char字段的值。 |
| 5 | abstract double get(String name, double val) 此方法从持久字段获取指定的double字段的值。 |
| 6 | abstract float get(String name, float val) 此方法从持久字段获取指定浮点(float)字段的值。 |
| 7 | abstract int get(String name, int val) 此方法从持久字段获取指定的整型(int)字段的值。 |
| 8 | abstract long get(String name, long val) 此方法从持久字段获取指定long字段的值。 |
| 9 | abstract Object get(String name, Object val) 此方法从持久字段获取指定的对象字段的值。 |
| 10 | abstract short get(String name, short val) 此方法从持久字段获取指定的short字段的值。 |
| 11 | abstract ObjectStreamClass getObjectStreamClass() 此方法获取描述流中的ObjectStreamClass字段。 |

# 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.ObjectOutputStream类 - Java.io包

**Java.io.ObjectOutputStream** 类写入基本数据类型和Java对象的图形到OutputStream。这些对象可以被读取(重组)使用ObjectInputStream。

## 类声明

以下是java.io.ObjectOutputStream类的声明：

```
public class ObjectOutputStream
  extends OutputStream
     implements ObjectOutput, ObjectStreamConstants
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected ObjectOutputStream()** 这提供了一种方法，子类是完全重新实现ObjectOutputStream来不必分配仅由这个实现ObjectOutputStream的私有数据。 |
| 2 | **ObjectOutputStream(OutputStream out)** 这将创建一个指定的OutputStream写入到一个ObjectOutputStream。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | protected void annotateClass(Class <?> cl) 子类可以实现此方法允许类的数据被存储在流中。 |
| 2 | protected void annotateProxyClass(Class<?> cl) 子类可以实现此方法来存储自定义数据的描述符动态代理类的流中。 |
| 3 | void close() 此方法关闭该流。 |
| 4 | void defaultWriteObject() 此方法写入当前类的非静态和非瞬态字段写入此流。 |
| 5 | protected void drain() 此方法排出ObjectOutputStream的所有缓冲的数据。 |
| 6 | protected boolean enableReplaceObject(boolean enable) 此方法使流 |

| | |
|---|---|
| 6 | 执行流中的替换对象。 |
| 7 | void flush() 此方法刷新流。 |
| 8 | ObjectOutputStream.PutField putFields() 此方法检索用于缓冲持久性字段被写入流的对象。 |
| 9 | protected Object replaceObject(Object obj) 此方法允许ObjectOutputStream的受信任子类的序列化过程中一个对象替代另一个。 |
| 10 | void reset() 此方法复位将忽略已经写入流中的任何对象的状态。 |
| 11 | void useProtocolVersion(int version) 将数据写入流时，此方法指定流协议版本才能使用。 |
| 12 | void write(byte[] buf) 此方法写入的字节数组.. |
| 13 | void write(byte[] buf, int off, int len) 此方法写入的字节子数组。 |
| 14 | void write(int val) 此方法写入一个字节。 |
| 15 | void writeBoolean(boolean val) 此方法写入一个布尔值。 |
| 16 | void writeByte(int val) 此方法写入一个8位字节。 |
| 17 | void writeBytes(String str) 此方法写入一个String作为一个字节序列。 |
| 18 | void writeChar(int val) 此方法写入一个16位字符。 |
| 19 | void writeChars(String str) 此方法写入一个字符串作为字符的序列。 |
| 20 | protected void writeClassDescriptor(ObjectStreamClass desc) 此方法写入指定的类描述符的对象输出流。 |
| 21 | void writeDouble(double val) 此方法写入一个64位double。 |
| 22 | void writeFields() 此方法写入缓冲字段的流.. |
| 23 | void writeFloat(float val) 此方法写入一个32位浮点数。 |
| 24 | void writeInt(int val) 此方法写入一个32位整数。 |
| 25 | void writeLong(long val) 此方法写入一个64位long。 |
| 26 | void writeObject(Object obj) 此方法将指定的对象写入ObjectOutputStream中。 |
| 27 | protected void writeObjectOverride(Object obj) 此方法由子类重写默认writeObject方法。 |
| 28 | void writeShort(int val) 此方法写入一个16位的short.. |
| 29 | protected void writeStreamHeader() 提供此方法，所以子类可以追加或预先准备自己的头流。 |

| | |
|---|---|
| 30 | 流。 |
| 31 | void writerUTF(String str) 这个字符串在经修订的UTF-8格式此方法原始数据写入。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.ObjectOutputStream.PutField类 - Java.io包

**Java.io.ObjectOutputStream.PutField** 类提供编程访问持久字段写入ObjectOutput。

## 类声明

以下是Java.io.ObjectOutputStream.PutField类的声明：

```
public abstract static class ObjectOutputStream.PutField
  extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **ObjectOutputStream.PutField()** 单一构造函数 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | abstract void put(String name, boolean val) 此方法把指定的布尔字段的值放到持久性字段。 |
| 2 | abstract void put(String name, byte val) 此方法把指定的字节字段的值放到持久性字段。 |
| 3 | abstract void put(String name, char val) 此方法把指定的char字段的值放到持久性字段。 |
| 4 | abstract void put(String name, double val) 此方法把指定的双字段的值放到持久性字段。 |
| 5 | abstract void put(String name, float val) 此方法把指定的浮点型字段的值放到持久性字段。 |
| 6 | abstract void put(String name, int val) 此方法把指定的整型字段的值放到持久性字段。 |
| 7 | abstract void put(String name, long val) 此方法把指定的long字段的值放到持久性字段。 |
| 8 | abstract void put(String name, Object val) 此方法把指定的对象字段的值放到持久性字段。 |
| 9 | abstract void put(String name, short val) 此方法把指定的短字段的值放到持久性字段。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.ObjectStreamClass类 - Java.io包

**Java.io.ObjectStreamClass** 类是序列化的描述符类。它包含类的名称和serialVersionUID。ObjectStreamClass在这个Java虚拟机加载一个特定的类都可以使用查找方法找到/创建。

## 类声明

以下是java.io.ObjectStreamClass类的声明：

```
public class ObjectStreamClass
  extends Object
    implements Serializable
```

## 字段域

以下是java.io.ObjectStreamClass类的字段：

- static ObjectStreamField[] NO_FIELDS -- 这是serialPersistentFields值，表示不序列化字段。

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Class<?> forClass() 此方法返回这个版本被映射到类中的本地VM。 |
| 2 | ObjectStreamField getField(String name) 该方法通过名字得到这个类的字段。 |
| 3 | ObjectStreamField[] getFields() 此方法返回当前序列化类的字段的数组。 |
| 4 | String getName() 此方法返回由这个描述符描述的类的名称。 |
| 5 | long getSerialVersionUID() 此方法返回这个类的serialVersionUID。 |
| 6 | static ObjectStreamClass lookup(Class<?> cl) 此方法找到该描述符为可序列化的类。 |
| 7 | static ObjectStreamClass lookupAny(Class<?> cl) 此方法返回的描述符的任何类，不管它是否实现Serializable接口。 |
| 8 | String toString() 此方法返回描述这个ObjectStreamClass的字符串。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.ObjectStreamField类 - Java.io包

**Java.io.ObjectStreamField**类是可序列化字段来自Serializable类的描述。ObjectStreamFields数组用来声明一个类的序列化字段。

## 类声明

以下是Java.io.ObjectStreamField类的声明：

```
public class ObjectStreamField
  extends Object
    implements Comparable<Object>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **ObjectStreamField(String name, Class<?> type)** 使用指定类型创建一个序列化的字段。 |
| 2 | **ObjectStreamField(String name, Class<?> type, boolean unshared)** 这将创建ObjectStreamField表示给定的名称和类型可序列化字段。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int compareTo(Object obj) 这种方法此字段与其他ObjectStreamField比较。 |
| 2 | String getName() 此方法获取该字段的名称。 |
| 3 | int getOffset() 此方法返回字段的实例数据的偏移量。 |
| 4 | Class<?> getType() 此方法获取字段的类型。 |
| 5 | char getTypeCode() 此方法返回字段类型的字符编码。 |
| 6 | String getTypeString() 此方法返回JVM类型签名。 |
| 7 | boolean isPrimitive() 如果这个字段为基本类型，此方法返回true。 |
| 8 | boolean isUnshared() 此方法返回布尔值指示是否通过ObjectStreamField实例所表示的序列化字段是独享的。 |
| 9 | protected void setOffset(int offset) 此方法返回的实例数据中的偏移量。 |
| 10 | String toString() 此方法返回一个描述此字段的字符串。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Object

# Java.io.OutputStreamWriter类 - Java.io包

**Java.io.OutputStreamWriter** 类是从字符流桥接字节流。写入字符编码成使用指定的字符集的字节。

## 类声明

以下是Java.io.OutputStreamWriter类的声明：

```
public class OutputStreamWriter
  extends Writer
```

## 字段

以下是Java.io.OutputStreamWriter类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **OutputStreamWriter(OutputStream out)** 这将创建一个使用默认的字符编码的OutputStreamWriter。 |
| 2 | **OutputStreamWriter(OutputStream out, Charset cs)** 这将创建一个使用给定字符集的OutputStreamWriter。 |
| 3 | **OutputStreamWriter(OutputStream out, CharsetEncoder enc)** 这将创建一个使用给定的charset编码器的OutputStreamWriter。 |
| 4 | **OutputStreamWriter(OutputStream out, String charsetName)** 这将创建一个使用指定的字符集的OutputStreamWriter。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭该流，但要先刷新它。 |
| 2 | void flush() 此方法刷新流。 |
| 3 | String getEncoding() 此方法返回正在使用此流的字符编码的名称。 |
| 4 | void write(char[] cbuf, int off, int len) 此方法写入字符数组的一部分。 |
| 5 | void write(int c) 此方法写入的单个字符。 |
| 6 | void write(String str, int off, int len) 此方法将写入一个字符串的一部分。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Writer
- Java.io.Object

# Java.io.PipedInputStream类 - Java.io包

**Java.io.PipedInputStream**类是一个管道输入流，可以连接到一个管道输出流，管道输入流提供的所有数据字节写入管道输出流。以下是有关PipedInputStream的要点：

- 该管道输入流包含一个缓冲区，解耦读操作和写操作限制之内。
- 不推荐试图从单个线程使用这两个对象，因为它可能会死锁的线程。
- 管道，如果被提供的数据字节连接的管道输出流线程不再处于活动状态被打破。

## 类声明

以下是java.io.PipedInputStream类的声明：

```
public class PipedInputStream
  extends InputStream
```

## 字段

以下是java.io.PipedInputStream类中的字段：

- protected byte[] buffer -- 一个循环缓冲区在其中输入数据被放置。
- protected int in -- 在循环缓冲区的数据的下一个字节从连接的管道输出流中接收时将存储的位置的索引。
- protected int out -- 在循环缓冲区的位置，在该数据的下一个字节将通过这个管道输入流中读取的索引。
- protected static int PIPE_SIZE -- 管道的循环输入缓冲区的默认大小。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **PipedInputStream()** 这创建了一个管道输入流，以便它尚未连接。 |
| 2 | **PipedInputStream(int pipeSize)** 这创建了一个管道输入流，以便它尚未连接，并使用指定的管道大小为管道缓冲区。 |
| 3 | **PipedInputStream(PipedOutputStream src)** 这将创建一个管道输入流，使其连接到管道输出流src中。 |
| 4 | **PipedInputStream(PipedOutputStream src, int pipeSize)** 这创建了一个管道输入流，以便它被连接到管道输出流src和使用指定的管道尺寸为管道缓冲区。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 此方法返回可以从此输入流中可无阻塞读取的字节数。 |
| 2 | void close() 此方法关闭此管道输入流并释放与该流关联的所有系统资源。 |
| 3 | void connect(PipedOutputStream src) 此方法使该管道输入流连接到管道输出流src中。 |
| 4 | int read() 此方法从这个管道输入流中读取下一个数据字节。 |
| 5 | int read(byte[] b, int off, int len) 此方法读取了从该管道输入流中len个字节数据到一个字节数组。 |
| 6 | protected void receive(int b) 这个方法接收一个字节的数据。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.InputStream
- Java.io.Object

# Java.io.PipedInputStream.available()方法实例 - Java.io包

**java.io.PipedInputStream.available()** 方法返回可以从此输入流中可无阻塞读取的字节数。

## 声明

以下是java.io.PipedInputStream.available()方法的声明

```
public int available()
```

## 参数

- NA

## 返回值

此方法返回可以从此输入流而不阻塞，或0读取的字节数，如果这个输入流已关闭通过调用close()方法，或者如果管道未连接，或中断。

## 异常

- IOException -- 如果发生I/ O错误。

## 例子

下面的示例演示java.io.PipedInputStream.available()方法的用法。

```
package com.yiibai;

import java.io.*;

public class PipedInputStreamDemo {

   public static void main(String[] args) {

      // create a new Piped input and Output Stream
      PipedOutputStream out = new PipedOutputStream();
      PipedInputStream in = new PipedInputStream();

      try {
         // connect input and output
         in.connect(out);

         // write something
         out.write(70);
         out.write(71);

         // print how many bytes are available
         System.out.println("" + in.available());

         // read what we wrote
         for (int i = 0; i < 2; i++) {
            System.out.println("" + (char) in.read());
         }

      } catch (IOException ex) {
         ex.printStackTrace();
      }

   }
}
```

让我们编译和运行上面的程序，这将产生以下结果：

```
2
F
G
```

# Java.io.PipedOutputStream类 - Java.io包

**Java.io.PipedOutputStream** 类是一个管道输出流可以连接到一个管道输入流，以创建通信管道。以下是有关PipedOutputStream要点：

- 该管道输出流是管道的发送端。
- 不推荐试图从单个线程使用这两个对象，因为它可能会死锁的线程。
- 数据被写入到一个管道输出流对象由一个线程和数据是从由其他线程所连接的PipedInputStream读取。
- 该管道如果是从连接中读取数据字节的线程管道输入流被中断已不再是活动。

## 类声明

以下是java.io.PipedOutputStream类的声明：

```
public class PipedOutputStream
  extends OutputStream
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **PipedOutputStream()** 这将创建一个尚未连接到一个管道输入流的管道输出流。 |
| 2 | **PipedOutputStream(PipedInputStream snk)** 这将创建连接到指定的管道输入流的管道输出流。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭此管道输出流并释放与此流有关的所有系统资源。 |
| 2 | void connect(PipedInputStream snk) 此方法这个管道输出流连接到一个接收器。 |
| 3 | void flush() 此方法刷新此输出流并强制将所有缓冲的输出字节被写出。 |
| 4 | void write(byte[] b, int off, int len) 此方法从指定的字节数组开始到这个管道输出流偏移量off写入len字节。 |
| 5 | void write(int b) 此方法写入指定的字节到管道输出流。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.OutputStream
- Java.io.Object

# Java.io.PipedReader类 - Java.io包

**Java.io.PipedReader** 类是管道字符输入流。

## 类声明

以下是java.io.PipedReader类的声明：

```
public class PipedReader
  extends Reader
```

## 字段

以下是java.io.PipedReader类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **PipedReader()** 这创造了一个PipedReader以便它尚未连接。 |
| 2 | **PipedReader(int pipeSize)** 这将创建一个PipedReader以便它尚未连接，并使用指定的管道大小，管道缓冲区。 |
| 3 | **PipedReader(PipedWriter src)** 这将创建一个PipedReader以便它被连接到传送writer src中。 |
| 4 | **PipedReader(PipedWriter src, int pipeSize)** 这将创建一个PipedReader以便它被连接到传送writer src和使用指定的管道大小，管道缓冲区。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | void close() 此方法关闭这个管道流并释放与该流关联的所有系统资源。 |
| 2 | void connect(PipedWriter src) 此方法使该管道读取器连接到传送writer src中。 |
| 3 | int read() 这个方法从这个管道流中读取数据的下一个字符。 |
| 4 | int read(char[] cbuf, int off, int len) 此方法读取最多从这个管道流中len个数据字符转换成字符数组。 |
| 5 | boolean ready() 此方法判断此流是否已准备好被读取。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Reader
- Java.io.Object

# Java.io.PipedWriter类 - Java.io包

**Java.io.PipedWriter** 类是管道字符输出流。

## 类声明

以下是java.io.PipedWriter类的声明：

```
public class PipedWriter
  extends Writer
```

## 字段域

以下是java.io.PipedWriter类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **PipedWriter()** 这将创建一个尚未连接到一个管道读取一个管道写入器。 |
| 2 | **PipedWriter(PipedReader snk)** 这将创建一个写入器管道连接到指定的管道读取器。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭此管道输出流并释放与此流有关的所有系统资源。 |
| 2 | void connect(PipedReader snk) 此方法连接该管道写入器到一个接收器.. |
| 3 | void flush() 此方法刷新此输出流并强制写出所有缓冲的输出字符。 |
| 4 | void write(char[] cbuf, int off, int len) 此方法写入len个字符从指定的字符数组开始到这个管道输出流偏移量。 |
| 5 | void write(int c) 此方法写入指定的字符到管道输出流。 |

## 继承的方法

这个类继承自以下类方法：

- Java.io.Writer
- Java.io.Object

# Java.io.PrintStream类 - Java.io包

**Java.io.PrintStream**类添加功能到另一个输出流，打印各种数据值表示形式方便的能力。

## 类声明

以下是java.io.PrintStream类的声明：

```
public class PrintStream
  extends FilterOutputStream
    implements Appendable, Closeable
```

## 字段域

以下是java.io.PrintStream类中的字段：

- protected OutputStream out -- 输出流进行过滤。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | PrintStream(File file) 这将创建一个新的打印流，与指定的文件无自动行刷新。 |
| 2 | PrintStream(File file, String csn) 这将创建一个新的打印流，无自动行刷新，指定文件和字符集。 |
| 3 | PrintStream(OutputStream out) 这将创建一个新的打印流。 |
| 4 | PrintStream(OutputStream out, boolean autoFlush) 这将创建一个新的打印流。 |
| 5 | PrintStream(OutputStream out, boolean autoFlush, String encoding) 这将创建一个新的打印流。 |
| 6 | PrintStream(String fileName) 这将创建一个新的打印流，无自动行刷新的，与指定的文件名。 |
| 7 | PrintStream(String fileName, String csn) 这将创建一个新的打印流，无自动行刷新，用指定的文件名和字符集。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | PrintStream append(char c) 此方法将指定字符追加到当前输出流。 |
| 2 | PrintStream append(CharSequence csq) 此方法将指定的字符序列此输出流。 |
| 3 | PrintStream append(CharSequence csq, int start, int end) 此方法将指定的字符序列的子序列来此输出流。 |
| 4 | boolean checkError() 此方法刷新流并检查其错误状态。 |
| 5 | protected void clearError() 此方法清除该流的内部错误状态。 |
| 6 | void close() 此方法关闭该流。 |
| 7 | void flush() 此方法刷新流。 |
| 8 | PrintStream format(Locale l, String format, Object... args) 此方法写入一个格式化字符串使用指定格式字符串和参数此输出流。 |
| 9 | PrintStream format(String format, Object... args) 此方法写入一个格式化字符串使用指定格式字符串和参数此输出流。 |
| 10 | void print(boolean b) 此方法打印一个布尔值。 |
| 11 | void print(char c) 这种方法打印的字符。 |
| 12 | void print(char[] s) 这种方法打印一个字符数组。 |
| 13 | void print(double d) 此方法打印一个双精度浮点数。 |
| 14 | void print(float f) 此方法打印一个浮点数。 |
| 15 | void print(int i) 此方法打印一个整数-int。 |
| 16 | void print(long l) 此方法打印一个长整数-long。 |
| 17 | void print(Object obj) 此方法打印一个对象。 |
| 18 | void print(String s) 此方法打印一个字符串。 |
| 19 | PrintStream printf(Locale l, String format, Object... args) 这是一个方便的方法使用指定格式字符串和参数将格式化字符串写入此输出流。 |
| 20 | PrintStream printf(String format, Object... args) 这是一个方便的方法使用指定格式字符串和参数将格式化字符串写入此输出流。 |
| 21 | void println() 此方法通过写入行分隔符字符串终止当前行。 |
| 22 | void println(boolean x) 此方法打印一个布尔值，然后终止该行。 |
| 23 | void println(char x) 此方法打印一个字符，然后终止该行。 |
| | |

| | |
|---|---|
| 24 | void println(char[] x) 此方法打印字符数组，然后终止该行。 |
| 25 | void println(double x) 此方法打印double，然后终止该行。 |
| 26 | void println(float x) 此方法打印一个float，然后终止该行。 |
| 27 | void println(int x) 此方法打印一个整数int，然后终止该行。 |
| 28 | void println(long x) 此方法打印一个long，然后终止该行。 |
| 29 | void println(Object x) 此方法打印一个对象，然后终止该行。 |
| 30 | void println(String x) 此方法打印一个String，然后终止该行。 |
| 31 | protected void setError() 此方法设置流的错误状态为true。 |
| 32 | void write(byte[] buf, int off, int len) 此方法从指定的字节数组开始，在这个流offset偏移写入len字节。 |
| 33 | void write(int b) 此方法将指定字节写入此流。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.FilterOutputStream
- Java.io.Object

# Java.io.PrintWriter类 - Java.io包

Java.io.PrintWriter 类打印格式化对象的表示到文本输出流。

## 类声明

以下是java.io.PrintWriter类的声明：

```
public class PrintWriter
  extends Writer
```

## 字段域

以下是java.io.PrintWriter类中的字段：

- protected Writer out -- 这是该PrintWriter字符输出流。
- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **PrintWriter(File file)** 这将创建一个新的PrintWriter，不带自动行刷新指定文件。 |
| 2 | **PrintWriter(File file, String csn)** 这将创建一个新的PrintWriter，不带自动行刷新的指定文件和字符集。 |
| 3 | **PrintWriter(OutputStream out)** 这将创建一个新的PrintWriter，无自动行刷新，从现有的OutputStream。 |
| 4 | **PrintWriter(OutputStream out, boolean autoFlush)** 这将创建从现有的OutputStream一个新的PrintWriter。 |
| 5 | **PrintWriter(String fileName)** 这将创建一个新的PrintWriter，无自动行刷新，用指定的文件名。 |
| 6 | **PrintWriter(String fileName, String csn)** 这将创建一个新的PrintWriter，无自动行刷新，用指定的文件名和字符集。 |
| 7 | **PrintWriter(Writer out)** 这将创建一个新的PrintWriter，无自动行刷新。 |
| 8 | **PrintWriter(Writer out, boolean autoFlush)** 这将创建一个新的PrintWriter。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | PrintWriter append(char c) 此方法将指定字符追加到这个writer。 |
| 2 | PrintWriter append(CharSequence csq) 此方法将指定的字符序列到此writer。 |
| 3 | PrintWriter append(CharSequence csq, int start, int end) 此方法将指定的字符序列的子序列写入此writer。 |
| 4 | boolean checkError() 这个方法刷新流，如果它没有关闭，并检查其错误状态。 |
| 5 | protected void clearError() 这种方法清除该流的错误状态.. |
| 6 | void close() 此方法关闭该流并释放与之关联的所有系统资源。 |
| 7 | void flush() 此方法刷新该流.. |
| 8 | PrintWriter format(Locale l, String format, Object... args) 使用指定的格式字符串和参数此方法写入一个格式化的字符串写入此writer.. |
| 9 | PrintWriter format(String format, Object... args) 使用指定的格式字符串和参数此方法写入一个格式化的字符串写入此writer.. |

| | |
|---|---|
| 10 | void print(boolean b) 此方法打印一个布尔值。 |
| 11 | void print(char c) 此方法打印的字符。 |
| 12 | void print(char[] s) 此方法打印一个字符数组.. |
| 13 | void print(double d) 此方法打印一个双精度浮点数。 |
| 14 | void print(float f) 此方法打印一个浮点数。 |
| 15 | void print(int i) 此方法打印一个整数。 |
| 16 | void print(long l) 此方法打印一个长整数。 |
| 17 | void print(Object obj) 此方法打印对象。 |
| 18 | void print(String s) 此方法打印一个字符串。 |
| 19 | PrintWriter printf(Locale l, String format, Object... args) 这是一个方便的方法使用指定格式字符串和参数将格式化字符串写入此writer中。 |
| 20 | PrintWriter printf(String format, Object... args) 这是一个方便的方法使用指定格式字符串和参数将格式化字符串写入此writer中。 |
| 21 | void println() 此方法通过写入行分隔符字符串终止当前行。 |
| 22 | void println(boolean x) 此方法打印一个布尔值，然后终止该行。 |
| 23 | void println(char x) 此方法打印一个字符，然后终止该行。 |
| 24 | void println(char[] x) 此方法将打印的字符数组，然后终止该行。 |
| 25 | void println(double x) 此方法打印一个双精度浮点数，然后终止该行。 |
| 26 | void println(float x) 此方法打印一个浮点数，然后终止该行。 |
| 27 | void println(int x) 此方法将打印的整数，然后终止该行。 |
| 28 | void println(long x) 此方法打印一个长整数，然后终止该行。 |
| 29 | void println(Object x) 此方法打印一个对象，然后终止该行。 |
| 30 | void println(String x) 此方法打印一个String，然后终止该行 |
| 31 | protected void setError() 此方法表明发生了错误。 |
| 32 | void write(char[] buf) 此方法写入字符数组。 |
| 33 | void write(char[] buf, int off, int len) 此方法写入字符数组的一部分。 |
| 34 | void write(int c) 此方法写入的单个字符。 |
| 35 | void write(String s) 此方法写入一个字符串。 |
| 36 | void write(String s, int off, int len) 此方法将一个字符串的一部分写入。 |

# 方法继承

这个类从以下类继承的方法：

- Java.io.Object

# Java.io.PushbackInputStream类 - Java.io包

**Java.io.PushbackInputStream**类添加功能到另一个输入流，即“push back”或“未unread”一个字节。

## 类声明

以下是java.io.PushbackInputStream类的声明：

```
public class PushbackInputStream
   extends FilterInputStream
```

## 字段域

以下是java.io.PushbackInputStream类中的字段：

- protected byte[] buf -- 推回缓冲区。
- protected int pos -- 推回缓冲区从下一个字节将被读取中的位置。
- protected InputStream in -- 将输入流进行过滤。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **PushbackInputStream(InputStream in)** 这将创建一个PushbackInputStream并保存其参数，在输入流中，以备后用。 |
| 2 | **PushbackInputStream(InputStream in, int size)** 这将创建具有指定大小的推回缓冲区PushbackInputStream，并保存其参数，在输入流中，以备后用。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 这个方法从这个输入流中可通过一个方法的下一次调用阻塞该输入流返回可以读取(或跳过)的字节数的估计值。 |
| 2 | void close() 此方法关闭此输入流并释放与该流关联的所有系统资源。 |
| 3 | void mark(int readlimit) 该方法标志着在此输入流的当前位置。 |
| 4 | boolean markSupported() 如果此输入流是否支持mark和reset方法，它不此方法测试。 |
| 5 | int read() 此方法读取从这个输入流数据的下一个字节。 |
| 6 | int read(byte[] b, int off, int len) 此方法从这个输入流中读取len个数据的字节到字节数组。 |
| 7 | void reset() 这个方法重新定位此流，以当时的mark方法最后调用这个输入流中的位置。 |
| 8 | long skip(long n) 这种方法跳过并丢弃n个字节从此输入流中的数据。 |
| 9 | void unread(byte[] b) 这种方法推回的字节将其复制到推回缓冲区前面的数组。 |
| 10 | void unread(byte[] b, int off, int len) 该方法通过将其复制到推回缓冲区的前面推回一个字节数组的一部分。 |
| 11 | void unread(int b) 此方法通过将其复制到推回缓冲区前面推回一个字节。 |

## 继承的方法

这个类从以下类继承的方法：

- Java.io.FilterInputStream
- Java.io.Object

# Java.io.RandomAccessFile类 - Java.io包

**Java.io.RandomAccessFile** 类文件的行为就像一个大数组存储在文件系统中的字节。这个类的实例支持读取和写入随机访问文件。

## 类声明

以下是java.io.RandomAccessFile类的声明：

```
public class RandomAccessFile
  extends Object
    implements DataOutput, DataInput, Closeable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **RandomAccessFile(File file, String mode)** 这将创建一个随机访问文件流来读取，并选择性地写入，由File参数指定的文件。 |
| 2 | **RandomAccessFile(File file, String mode)** 这将创建一个随机访问文件流来读取，并选择性地写入，具有指定名称的文件。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭此随机存取文件流并释放与该流关联的所有系统资源。 |
| 2 | FileChannel getChannel() 此方法返回与此文件关联的唯一文件通道对象。 |
| 3 | FileDescriptor getFD() 此方法返回与此流关联的不透明文件描述符对象。 |
| 4 | long getFilePointer() 此方法返回当前在此文件中的偏移。 |
| 5 | long length() 这个方法返回当前文件的长度。 |
| 6 | int read() 此方法读取数据从该文件一个字节。 |
| 7 | int read(byte[] b) 此方法读取为从该文件b.length个数据字节为字节数组。 |

| | |
|---|---|
| 8 | int read(byte[] b, int off, int len) 此方法读取为从该文件len个字节数据到一个字节数组。 |
| 9 | boolean readBoolean() 此方法读取该文件一个布尔值。 |
| 10 | byte readByte() 此方法从该文件读取有符号8位的值。 |
| 11 | char readChar() 此方法从文件读取一个字符。 |
| 12 | double readDouble() 此方法从文件读取一个double数。 |
| 13 | float readFloat() 此方法从文件读取一个浮点数。 |
| 14 | void readFully(byte[] b) 此方法读取该文件b.length个字节到字节数组，并从当前文件指针。 |
| 15 | void readFully(byte[] b, int off, int len) 此方法读取这个文件正好len个字节到字节数组，并从当前文件指针。 |
| 16 | int readInt() 此方法从该文件中读取一个有符号的32位整数。 |
| 17 | String readLine() 此方法从该文件中读取文本的下一行。 |
| 18 | long readLong() 此方法从该文件中读取一个有符号的64位整数。 |
| 19 | short readShort() 此方法从该文件中读取一个有符号的16位数。 |
| 20 | int readUnsignedByte() 此方法从该文件中读取一个无符号的八位数。 |
| 21 | int readUnsignedShort() 此方法从该文件中读取一个无符号的16位数。 |
| 22 | String readUTF() 从这个文件中的字符串此方法读取。 |
| 23 | void seek(long pos) 此方法设置文件指针偏移量，从这个文件开始测量，进行下一个读或写操作发生。 |
| 24 | void setLength(long newLength) 此方法设置此文件的长度。 |
| 25 | int skipBytes(int n) 此方法尝试跳过n个字节的输入丢弃跳过的字节。 |
| 26 | void write(byte[] b) 此方法写入b.length个字节从指定的字节数组到该文件，并从当前文件指针。 |
| 27 | void write(byte[] b, int off, int len) 此方法从指定的字节数组开始到该文件偏移量off写入len字节。 |
| 28 | void write(int b) 此方法写入指定的字节写入此文件。 |
| 29 | void writeBoolean(boolean v) 此方法写入一个布尔值，该文件为一个字节的值。 |
| 30 | void writeByte(int v) 此方法写入一个字节到文件作为一个单字节值。 |
| 31 | void writeBytes(String s) 此方法写入字符串到文件为一个字节序列。 |
| 32 | void writeChar(int v) 此方法写入一个字符的文件作为一个双字节值，高字节在前。 |

| | |
|---|---|
| 33 | void writeChars(String s) 此方法将一个字符串写入该文件作为一个字符序列。 |
| 34 | void writeDouble(double v) 此方法double参数转换为long使用doubleToLongBits方法在类Double，然后写到long值的文件作为八字节数量，高字节在前。 |
| 35 | void writeFloat(float v) 此方法float参数转换为使用floatToIntBits方法在类Float一个int，然后写到int值，以该文件为一个四字节数量，高字节在前。 |
| 36 | void writeInt(int v) 此方法写入一个int到文件为四个字节，高字节在前。 |
| 37 | void writeLong(long v) 此方法写入一个长的文件作为八个字节，高字节在前。 |
| 38 | void writeShort(int v) 此方法写入一个短的文件为两个字节，高字节在前。 |
| 39 | void writeUTF(String str) 这种方法将一个字符串写入使用经修订的UTF-8编码以与机器无关的方式的文件。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.Object

# Java.io.Reader类 - Java.io包

Java.io.Reader 类是一个抽象类，用于读取字符流。

## 类声明

以下是java.io.Reader类的声明：

```
public class Reader
  extends Object
    implements DataOutput, DataInput, Closeable
```

## 字段域

以下是java.io.Reader类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **protected Reader()** 这将创建一个新的字符流reader，其重要部分将在读取器本身同步。 |
| 2 | **protected Reader(Object lock)** 这将创建一个新的字符流reader，其重要部分将在给定的对象同步。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | abstract void close() 此方法关闭该流并释放与之关联的所有系统资源。 |
| 2 | void mark(int readAheadLimit) 这种方法标记流中的当前位置。 |
| 3 | boolean markSupported() 这个方法告诉此流是否支持mark()操作。 |
| 4 | int read() 此方法读取单个字符。 |
| 5 | int read(char[] cbuf) 此方法读取字符到一个数组中。 |
| 6 | abstract int read(char[] cbuf, int off, int len) 此方法读取字符到一个数组中的一部分。 |
| 7 | int read(CharBuffer target) 此方法试图读取字符入指定的字符缓冲区。 |
| 8 | boolean ready() 此方法通知此流是否已准备好被读取。 |
| 9 | void reset() 这种方法重置流。 |
| 10 | long skip(long n) 此方法跳过字符。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.Object

# Java.io.SequenceInputStream类 - Java.io包

**Java.io.SequenceInputStream** 类代表的其他输入流的逻辑连接。它开始时与输入流的有序集合，并从第一个到文件结束读取为止，之后将其从第二个读出，依此类推，直到文件的末尾的最后的包含的输入流。

## 类声明

以下是java.io.SequenceInputStream类的声明：

```
public class SequenceInputStream
  extends InputStream
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **SequenceInputStream(Enumeration<? extends InputStream> e)** 这将初始化通过记住参数，它必须是生产对象的运行时类型为InputStream枚举一个新创建的SequenceInputStream. |
| 2 | **SequenceInputStream(InputStream s1, InputStream s2)** 这将初始化通过记住两个参数，这将是按顺序阅读，第一S1和S2，然后，提供从这个SequenceInputStream读取的字节一个新创建的SequenceInputStream。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 这个方法从当前底层输入流中可通过方法下一次调用阻塞当前底层输入流返回可以读取(或跳过)的字节数的估计值。 |
| 2 | void close() 此方法关闭此输入流并释放与该流关联的所有系统资源。 |
| 3 | int read() 此方法读取从这个输入流数据的下一个字节。 |
| 4 | int read(byte[] b, int off, int len) 此方法读取最多len个从这个输入流中数据的字节到字节数组。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.InputStream
- Java.io.Object

# Java.io.SerializablePermission类 - Java.io包

**Java.io.SerializablePermission**类是可序列化的权限。SerializablePermission包含一个名称(也称为“target name”)，但没有动作列表;要么有命名的权限，也可以没有。目标名称是可序列化权限的名称。

## 类声明

以下是Java.io.SerializablePermission类的声明：

```
public final class SerializablePermission
  extends BasicPermission
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **SerializablePermission(String name)** 这将创建一个新的SerializablePermission具有指定名称。 |
| 2 | **SerializablePermission(String name, String actions)** 这将创建具有指定名称的新的SerializablePermission对象。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.BasicPermission
- Java.io.Permission
- Java.io.Object

# Java.io.StreamTokenizer类 - Java.io包

**Java.io.StreamTokenizer**类获取输入流并将其解析为"标记"，允许令牌被读取一次。流标记生成器可以识别标识符，数字，带引号的字符串，以及各种注释样式。

## 类声明

以下是java.io.StreamTokenizer类的声明：

```
public class StreamTokenizer
  extends Object
```

## 字段域

以下是java.io.StreamTokenizer类中的字段：

- double nval -- 如果当前标记是一个数字，此字段包含该数字的值。
- String sval -- 如果当前标记是一个文字标记，则此字段包含一个字符串，给出该文字标记的字符。
- static int TT_EOF -- 一个常量，表明流的末尾已被读取。
- static int TT_EOL -- 一个常数，指示末尾行已被读取。
- static int TT_NUMBER -- 一个常量，表示一个数字标记已读。
- static int TT_WORD -- 一个常量，表示一个文字标记已读。
- int ttype -- 调用nextToken方法后，此字段包含刚读取的标记的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **StreamTokenizer(Reader r)** 这将创建一个标记者，解析给定的字符流。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void commentChar(int ch) 指定的字符参数启动一个单行注释。 |
| 2 | void eolIsSignificant(boolean flag) 这种方法确定的线端部是否被视为令牌。 |
| 3 | int lineno() 此方法返回当前行号。 |
| 4 | void lowerCaseMode(boolean fl) 此方法确定是否文字标记会自动小写。 |
| 5 | int nextToken() 此方法分析从标记生成器的输入流中的下一个标记。 |
| 6 | void ordinaryChar(int ch) 这个方法指定了字符的参数是“ordinary”这个词器。 |
| 7 | void ordinaryChars(int low, int hi) 本方法规定，在范围内的所有字符C在low <= c <= high是“ordinary”这个词器。 |
| 8 | void parseNumbers() 这种方法指定数字应该由这个标记生成器解析。 |
| 9 | void pushBack() 这种方法会导致此标记生成器的nextToken方法的下一次调用返回的当前值在ttype字段和不修改该值在nval 或 sval 字段中。 |
| 10 | void quoteChar(int ch) 这种方法指定的匹配对这个人物的分隔字符串常量在此标记生成器. |
| 11 | void resetSyntax() 让所有的字符都是这种方法重置此标记生成器的语法表示“ordinary”。请参阅上一个字符为普通详细信息，此时要用ordinaryChar方法。 |
| 12 | void slashSlashComments(boolean flag) 此方法确定是否标记生成器识别的C + +风格的注释。 |
| 13 | void slashStarComments(boolean flag) 此方法确定是否标记生成器在识别C风格的注释。 |
| 14 | String toString() 此方法返回当前流标记的字符串表示形式，它发生在的行号。 |
| 15 | void whitespaceChars(int low, int hi) 这种方法指定的范围low <= c <= high 内为空白字符的所有字符角 |
| 16 | void wordChars(int low, int hi) 本方法规定，在范围内的所有字符C在low <= c >= high单词成分。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.Object

# Java.io.StringBufferInputStream类 - Java.io包

**Java.io.StringBufferInputStream**类允许应用程序创建，其中读取的字节由字符串的内容提供的输入流。应用程序还可以通过使用ByteArrayInputStream.Only字符串中的低八位每个字符都使用这个类读取一个字节数组字节。

这个此类已被Oracle否决，不再被使用。

## 类声明

以下是Java.io.StringBufferInputStream类的声明：

```
public class StringBufferInputStream
   extends InputStream
```

## 字段域

以下是Java.io.StringBufferInputStream类中的字段：

- protected String buffer -- 这是字节被读取的字符串。
- protected int count -- 这是在输入数据流缓存器有效字符数。
- protected int pos -- 这是下一个字符从输入流缓存器读取的索引。

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **StringBufferInputStream(String s)** 这将创建一个字符串输入流中读取指定的字符串数据。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int available() 此方法返回可以从输入流中可无阻塞读取的字节数。 |
| 2 | int read() 此方法读取从这个输入流数据的下一个字节。 |
| 3 | int read(byte[] b, int off, int len) 此方法读取最多len个从这个输入流中数据的字节到字节数组。 |
| 4 | void reset() 这种方法重置输入流，开始从这个输入流的基础缓冲区的第一个字符读。 |
| 5 | long skip(long n) 此方法跳过输入流中的n个字节。 |

# 方法继承

这个类从以下类继承的方法：

- Java.io.InputStreams
- Java.io.Object

# Java.io.StringReader类 - Java.io包

**Java.io.StringReader**类是字符流的源，是一个字符串。

## 类声明

以下是java.io.StringReader类的声明：

```
public class StringReader
  extends Reader
```

## 字段域

以下是java.io.StringReader类中的字段：

- protected Object lock -- 这是用于同步针对此流操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **StringReader(String s)** 这将创建一个新的字符串读取器。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭该流并释放与之关联的所有系统资源。 |
| 2 | void mark(int readAheadLimit) 这种方法标记流中的当前位置。 |
| 3 | boolean markSupported() 此方法通知此流是否支持mark()操作，这确实如此。 |
| 4 | int read() 此方法读取单个字符。 |
| 5 | int read(char[] cbuf, int off, int len) 此方法读取字符到一个数组中的一部分。 |
| 6 | boolean ready() 此方法通知此流是否已准备好被读取。 |
| 7 | void reset() 此方法重置流为最新的标记，或以该字符串的开头，如果它从来没有被标记。 |
| 8 | long skip(long ns) 此方法跳过流中指定的字符数。 |

# 方法继承

这个类从以下类继承的方法：

- Java.io.Reader
- Java.io.Object

# Java.io.Writer类 - Java.io包

**Java.io.Writer** 类是一个抽象类，用于写入字符流。

## 类声明

以下是java.io.Writer类的声明：

```
public abstract class Writer
  extends Object
    implements Appendable, Closeable, Flushable
```

## 字段域

以下是java.io.Writer类中的字段：

- protected Object lock -- 这是用于同步针对此流的操作的对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected Writer()** 这将创建一个新的字符流writer，其关键部分将在writer 本身同步。 |
| 2 | **protected Writer(Object lock)** 这将创建一个新的字符流writer，其关键部分将在给定的对象同步。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Writer append(char c) 此方法将指定字符追加到这个writer。 |
| 2 | Writer append(CharSequence csq) 此方法将指定的字符序列到这个writer。 |
| 3 | Writer append(CharSequence csq, int start, int end) 此方法将指定的字符序列的子序列写入此writer。 |
| 4 | abstract void close() 此方法丢失流，但要先刷新它。 |
| 5 | abstract void flush() 此方法刷新流。 |
| 6 | void write(char[] cbuf) 此方法写入字符数组。 |
| 7 | abstract void write(char[] cbuf, int off, int len) 此方法写入字符数组的一部分。 |
| 8 | void write(int c) 此方法写入的单个字符。 |
| 9 | void write(String str) 此方法写入一个字符串。 |
| 10 | void write(String str, int off, int len) 此方法将写入一个字符串的一部分。 |

## 方法继承

这个类从以下类继承的方法：

- Java.io.Object

# Java.io.Interfaces - Java.io包

**java.io.Interfaces** 提供系统输入和输出通过数据流，序列化和文件系统。

## 接口汇总

| S.N. | 接口与说明 |
|---|---|
| 1 | **Closeable** 这是一个源或数据的目的地，可以关闭。 |
| 2 | **DataInput** 这提供了从二进制流中读取字节，并从他们在任何Java基本类型的数据重建。 |
| 3 | **DataOutput** 这提供了将数据从任意Java基本类型的一系列字节并将其写入字节的二进制流。 |
| 4 | **Externalizable** 这仅提供该类一个Externalizable实例的标识被写入序列化流，它是类来保存和恢复其实例的内容。 |
| 5 | **FileFilter** 这是一个抽象路径名的过滤器。 |
| 6 | **FilenameFilter** 这是实现此接口是用来过滤文件名的类的实例。 |
| 7 | **Flushable** 这是数据的目的地，可以被刷新。 |
| 8 | **ObjectInput** 这扩展了的DataInput接口以包含对象的读取。 |
| 9 | **ObjectInputValidation** 这是回调接口，以允许一个图中的对象的验证。 |
| 10 | **ObjectOutput** 这是objectOutput扩展DataOutput中的接口，包括写对象。 |
| 11 | **ObjectStreamConstants** 写入的对象序列化流中的常量。 |
| 12 | **Serializable** 这是启用由类实现java.io.Serializable接口。 |

# java.lang

java.util包中包含有基本的Java编程语言程序设计的类。本教程引用将采用简单实用的例子引导您完成所有的java.lang包中可用的方法。

## 读者

本教程参考是为初学者准备，帮助他们了解相关的java.lang包中所有可用的方法的基本功能。

## 提前条件

通过做练习，拥有在此引用给定各种类的例子之前，假设你已经知道基本的Java编程。

# java.lang.Boolean类 - java.lang

**java.lang.Boolean** 类封装在一个对象基本布尔型的值。 Boolean类型的对象包含一个字段，它的类型是布尔值。

## 类声明

以下是 **java.lang.Boolean** 类的声明：

```
public final class Boolean
   extends Object
      implements Serializable, Comparable<Boolean>
```

## 字段域

以下是java.lang.Boolean类的字段：

- static Boolean FALSE -- 这是对应于原始的Boolean对象的值false。
- static Boolean TRUE -- 这是对应于Boolean对象的原始值true 。
- static Class<Boolean> TYPE -- 这是代表基本布尔型的Class对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Boolean(boolean value)** 这代表分配值参数的Boolean对象。 |
| 2 | **Boolean(String s)** 这表示分配Boolean对象的值为true，如果字符串参数不为null，等于忽略大小写的字符串“true”。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean booleanValue() 此方法将返回此布尔对象为布尔原始值。 |
| 2 | int compareTo(Boolean b) 此方法将此Boolean实例与另一个对象实例进行比较。 |
| 3 | boolean equals(Object obj) 当且仅当参数不为null，并且是一个布尔对象，表示同样的布尔值作为此对象时，此方法返回true。 |
| 4 | static boolean getBoolean(String name) 当且仅当存在以参数命名的系统属性和等于该字符串“true”此方法返回true。 |
| 5 | int hashCode() 此方法将返回这个布尔对象的哈希码。 |
| 6 | static boolean parseBoolean(String s) 此方法解析字符串参数为布尔值。 |
| 7 | String toString() 此方法将返回表示该布尔值的String对象。 |
| 8 | static String toString(boolean b) 此方法将返回一个表示指定布尔的String对象。 |
| 9 | static Boolean valueOf(boolean b) 此方法将返回一个表示指定布尔值的Boolean实例。 |
| 10 | static Boolean valueOf(String s) 此方法返回一个布尔与指定字符串表示的值。 |

# 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Byte类 - java.lang

**java.lang.Byte** 类包装基本类型字节的对象的值。 Byte类型的对象包含一个字段，它的类型是字节。

## 类声明

以下是**java.lang.Byte**类的声明：

```
public final class Byte
   extends Number
      implements Comparable<Byte>
```

## 字段域

以下是java.lang.Byte类中的字段：

- static byte MAX_VALUE -- 持有常数一个字节可以有最大值, 2<sup style="margin: 0px; padding: 0px; font-size: 13px;">7</sup>-1.
- static byte MIN_VALUE -- 此常数持有的一个字节的最小值 -2<sup style="margin: 0px; padding: 0px; font-size: 13px; background-color: initial;">7</sup>.
- static int SIZE -- 此是用来表示于二进制补码形式的字节值的比特数。
- static Class<Byte> TYPE -- 这是类实例，表示基本类型字节。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Byte(byte value)** 此构造一个新分配的字节对象，表示指定的字节值。 |
| 2 | **Byte(String s)** 此构造一个新分配的字节对象，表示String参数指定的字节值。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | byte byteValue() 此方法返回字节一个字节的值。 |
| 2 | int compareTo(Byte anotherByte) 此方法数值比较两个字节的对象。 |
| 3 | static Byte decode(String nm) 此方法解码字符串转换为字节。 |
| 4 | double doubleValue() 此方法将返回此字节为double值。 |
| 5 | boolean equals(Object obj) 此方法将此对象与指定对象比较。 |
| 6 | float floatValue() 此方法将返回此字节为float的值。 |
| 7 | int hashCode() 此方法将返回这个字节Byte的哈希码。 |
| 8 | int intValue() 此方法将返回此字节Byte为一个int值。 |
| 9 | long longValue() 此方法将返回此字节作为一个long的值。 |
| 10 | static byte parseByte(String s) 此方法将字符串参数作为符号的十进制字节。 |
| 11 | static byte parseByte(String s, int radix) 此方法解析字符串参数作为第二个参数指定的基数符号的字节。 |
| 12 | short shortValue() 此方法返回字节Byte的short的值。 |
| 13 | String toString() 此方法返回一个代表此字节的值的String对象。 |
| 14 | static String toString(byte b) 此方法将返回一个表示指定字节一个新的String对象。 |
| 15 | static Byte valueOf(byte b) 此方法将返回一个表示指定的字节值的字节实例。 |
| 16 | static Byte valueOf(String s) 此方法将返回一个字节对象持有指定的字符串中给定的值。 |
| 17 | static Byte valueOf(String s, int radix) 该方法返回一个字节的对象保持从指定的String中提取的值时，由第二个参数给出的基数进行分析。 |

# 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Character - java.lang

java.lang.Character 类封装在一个对象的基本类型char值。字符类型的对象包含一个字段，它的类型为char。

## 类声明

以下是java.lang.Character类的声明：

```
public final class Character
   extends Object
      implements Serializable, Comparable<Character>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **Character(char value)** 此构造一个新分配字符对象，表示指定的char值。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | static int charCount(int codePoint) 此方法确定表示指定字符所需的字符的值的数目(Unicode代码点)。 |
| 2 | char charValue() 此方法返回字符对象的值。 |
| 3 | static int codePointAt(char[] a, int index) 此方法返回char数组的给定索引上的代码点。 |
| 4 | static int codePointAt(char[] a, int index, int limit) 此方法返回的字符数组，只能用于数组元素与指数低于上限的定索引上的代码点。 |
| 5 | static int codePointAt(CharSequence seq, int index) 此方法返回CharSequence给定索引上的代码点。 |
| 6 | static int codePointBefore(char[] a, int index) 此方法返回char数组的给定索引前面的代码点。 |
| 7 | static int codePointBefore(char[] a, int index, int start) 此方法返回字符数组，其中仅可用于数组元素与指数大于或等于开始的给定索引前面的 |

| | |
|---|---|
| | 代码点。 |
| 8 | static int codePointBefore(CharSequence seq, int index) 此方法返回CharSequence给定索引前面的代码点。 |
| 9 | static int codePointCount(char[] a, int offset, int count) 该方法在char数组参数的一个子返回的Unicode代码点的数量 |
| 10 | static int codePointCount(CharSequence seq, int beginIndex, int endIndex) 该方法在指定的字符序列的文本范围返回的Unicode代码点的数量。 |
| 11 | int compareTo(Character anotherCharacter) 该方法在数字上比较两个字符的对象。 |
| 12 | static int digit(char ch, int radix) 此方法返回指定基数字符ch的数值。 |
| 13 | static int digit(int codePoint, int radix) 此方法返回指定基数指定字符（Unicode代码点）的数值。 |
| 14 | boolean equals(Object obj) 该方法此对象与指定对象比较 |
| 15 | static char forDigit(int digit, int radix) 此方法确定的指定基数的特定数字的字符表示。 |
| 16 | static byte getDirectionality(char ch) 此方法返回给定字符的Unicode的方向属性。 |
| 17 | static byte getDirectionality(int codePoint) 该方法返回给定字符（Unicode代码点）的Unicode方向属性。 |
| 18 | static int getNumericValue(char ch) 此方法返回指定的Unicode字符表示的int值。 |
| 19 | static int getNumericValue(int codePoint) 此方法返回int值指定的字符（Unicode代码点）表示。 |
| 20 | static int getType(char ch) 此方法返回一个值，表明一个字符的普通类。 |
| 21 | static int getType(int codePoint) 此方法返回一个值，表明一个字符的普通类。 |
| 22 | int hashCode() 此方法返回这个字符的哈希码。 |
| 23 | static boolean isDefined(char ch) 此方法确定字符是否为Unicode定义。 |
| 24 | static boolean isDefined(int codePoint) 该方法确定的字符（Unicode代码点）在Unicode中定义。 |
| 25 | static boolean isDigit(char ch) 此方法确定指定字符是否一个数字。 |
| 26 | static boolean isDigit(int codePoint) 此方法确定指定字符（Unicode代码点）是其中一位。 |

| | |
|---|---|
| 27 | static boolean isHighSurrogate(char ch) 该方法判断给定char值是高代理项代码单元（也称为主导代理项代码单元）。 |
| 28 | static boolean isIdentifierIgnorable(char ch) 此方法确定指定字符是否应该被视为一个可忽略的字符在Java标识符或Unicode标识符。 |
| 29 | static boolean isIdentifierIgnorable(int codePoint) 此方法确定指定字符（Unicode代码点）应被视为一个可忽略的字符在Java标识符或Unicode标识符。 |
| 30 | static boolean isISOControl(char ch) 此方法确定指定字符是否ISO控制字符。 |
| 31 | static boolean isISOControl(int codePoint) 此方法确定引用的字符（Unicode代码点）是一家通过ISO控制字符。 |
| 32 | static boolean isJavaIdentifierPart(char ch) 该方法确定指定的字符可能是Java标识符中除首字符以外的部分。 |
| 33 | static boolean isJavaIdentifierPart(int codePoint) 该方法确定的字符（Unicode代码点）可能是Java标识符中除首字符以外的部分。 |
| 34 | static boolean isJavaIdentifierStart(char ch) 此方法确定指定字符是否允许在Java标识符的第一个字符。 |
| 35 | static boolean isJavaIdentifierStart(int codePoint) 该方法确定的字符（Unicode代码点）是否允许在Java标识符的第一个字符。 |
| 36 | static boolean isLetter(char ch) 此方法确定指定字符是否一个字母。 |
| 37 | static boolean isLetter(int codePoint) 此方法确定指定字符（Unicode代码点）是一个字母。 |
| 38 | static boolean isLetterOrDigit(char ch) 此方法确定指定字符是字母或数字。 |
| 39 | static boolean isLetterOrDigit(int codePoint) 此方法确定指定字符（Unicode代码点）是一个字母或数字。 |
| 40 | static boolean isLowerCase(char ch) 此方法确定指定字符是否为小写字母。 |
| 41 | static boolean isLowerCase(int codePoint) 此方法确定指定字符（Unicode代码点）是否为小写字母。 |
| 42 | static boolean isLowSurrogate(char ch) 该方法判断给定char值是低代理项代码单元（也称为尾部代理项代码单元）。 |
| 43 | static boolean isMirrored(char ch) 此方法确定按照Unicode规范的字符是否为镜像。 |
| 44 | static boolean isMirrored(int codePoint) 此方法确定指定字符（Unicode代码点）是否按照Unicode规范镜像。 |
| | static boolean isSpaceChar(char ch) 此方法确定指定字符是否为 |

| | |
|---|---|
| | Unicode空白字符。 |
| 46 | static boolean isSpaceChar(int codePoint) 此方法确定指定字符（Unicode代码点）是一个Unicode空格字符。 |
| 47 | static boolean isSupplementaryCodePoint(int codePoint) 此方法确定指定字符（Unicode代码点）是否在补充字符范围。 |
| 48 | static boolean isSurrogatePair(char high, char low) 此方法确定指定对char值是否一个有效的代理对。 |
| 49 | static boolean isTitleCase(char ch) 此方法确定指定字符是否是首字母大写字符。 |
| 50 | static boolean isTitleCase(int codePoint) 此方法确定指定字符（Unicode代码点）是一个首字母大写字符。 |
| 51 | static boolean isUnicodeIdentifierPart(char ch) 此方法确定指定字符可以是Unicode标识符中除首字符以外的部分。 |
| 52 | static boolean isUnicodeIdentifierPart(int codePoint) 此方法确定指定字符（Unicode代码点）可能是Unicode标识符中除首字符以外的部分。 |
| 53 | static boolean isUnicodeIdentifierStart(char ch) 此方法确定指定字符是否是允许作为Unicode标识符的第一个字符。 |
| 54 | static boolean isUnicodeIdentifierStart(int codePoint) 此方法确定指定字符（Unicode代码点）是允许作为Unicode标识符的第一个字符。 |
| 55 | static boolean isUpperCase(char ch 此方法确定指定字符是否是大写字符。 |
| 56 | static boolean isUpperCase(int codePoint) 此方法确定指定字符（Unicode代码点）是一个大写字母。 |
| 57 | static boolean isValidCodePoint(int codePoint) 此方法确定指定的代码点是否为0x0000～0x10FFFF之间包容的范围内有效的Unicode代码点值。 |
| 58 | static boolean isWhitespace(char ch) 此方法确定指定字符是否是根据Java的空白符。 |
| 59 | static boolean isWhitespace(int codePoint) 此方法确定指定字符（Unicode代码点）是根据Java的空白符。 |
| 60 | static int offsetByCodePoints(char[] a, int start, int count, int index, int codePointOffset) 此方法返回从给定指数codePointOffset码点的偏移给定的字符子数组中的索引 |
| 61 | static int offsetByCodePoints(CharSequence seq, int index, int codePointOffset) 此方法返回给定字符序列从给定的指数codePointOffset码点偏移中的索引。 |
| 62 | static char reverseBytes(char ch) 此方法返回通过反转指定char值的字 |

| | |
|---|---|
| 62 | 节的顺序而获得的值。 |
| 63 | static char[] toChars(int codePoint) 该方法将指定字符（Unicode代码点）到其存储在一个字符数组的UTF-16表示形式。 |
| 64 | static int toChars(int codePoint, char[] dst, int dstIndex) 该方法将指定字符（Unicode代码点）其UTF-16表示形式。 |
| 65 | static int toCodePoint(char high, char low) 该方法指定的代理对到其增补代码点值转换。 |
| 66 | static char toLowerCase(char ch) 该方法的参数字符使用来自UnicodeData文件的大小写映射信息转换成小写。 |
| 67 | static int toLowerCase(int codePoint) 此方法转换的字符（Unicode代码点）参数使用来自UnicodeData文件的大小写映射信息为小写。 |
| 68 | String toString() 该方法返回一个代表该字符的值的String对象。 |
| 69 | static String toString(char c) 此方法返回一个表示指定字符的String对象。 |
| 70 | static char toTitleCase(char ch) 该方法的参数字符使用来自UnicodeData文件的大小写映射信息标题字符转换。 |
| 71 | static int toTitleCase(int codePoint) 此方法转换的字符（Unicode代码点）参数使用来自UnicodeData文件的大小写映射信息标题字符。 |
| 72 | static char toUpperCase(char ch) 该方法的参数字符使用来自UnicodeData文件的大小写映射信息转换为大写。 |
| 73 | static int toUpperCase(int codePoint) 此方法转换的字符（Unicode代码点）参数使用来自UnicodeData文件的大小写映射信息为大写。 |
| 74 | static Character valueOf(char c) 此方法返回一个表示指定字符值的字符实例。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Character.Subset类 - java.lang

**java.lang.Character.Subset** 类实例表示Unicode字符集的特定子集。在字符类中定义子集的唯一的关联关系是UnicodeBlock。

## 类声明

以下是**java.lang.Character.Subset**类的声明：

```
public static class Character.Subset
   extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected Character.Subset(String name)** 这构造了一个新的子集实例。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean equals(Object obj) 此方法比较两个子集的对象相等对。 |
| 2 | int hashCode() 此方法返回的是Object.hashCode()方法中定义的标准的散列码。 |
| 3 | String toString() 此方法返回该子集的名称。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Character.UnicodeBlock类 - java.lang

**java.lang.Character.UnicodeBlock** 类是一个代表Unicode规范的字符块字符子集。字符块通常定义用于特定的脚本或目的字符。

## 类声明

以下是**java.lang.Character.UnicodeBlock**类的声明：

```
public static final class Character.UnicodeBlock
   extends Character.Subset
```

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | static Character.UnicodeBlock forName(String blockName) 此方法返回具有指定名称的UnicodeBlock。 |
| 2 | static Character.UnicodeBlock of(char c) 此方法返回表示包含给定字符，则返回null如果字符不是定义块的成员的Unicode块的对象。 |
| 3 | static Character.UnicodeBlock of(int codePoint) 此方法返回表示包含给定字符(Unicode代码点)的Unicode块的对象，或者为null，如果该字符不是定义块的成员。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Character.Subset
- java.lang.Object

# java.lang.Class类 - java.lang

java.lang.Class 类的实例表示正在运行的Java应用程序中的类和接口。它没有公共的构造函数。

## 类声明

以下是java.lang.Class类的声明：

```
public final class Class<T>
  extends Object
    implements Serializable, GenericDeclaration, Type, AnnotatedEle
```

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | <U> Class<? extends U> asSubclass(Class<U> clazz) 此方法投射此Class对象，以表示指定的Class对象所表示的类的子类。 |
| 2 | T cast(Object obj) 此方法投射的目的是通过此Class对象所表示的类或接口。 |
| 3 | boolean desiredAssertionStatus() 此方法返回将被分配到这个类，如果它在调用此方法时被初始化的断言状态。 |
| 4 | static Class<?> forName(String className) 此方法返回的类或接口与给定的字符串名称相关联的Class对象。 |
| 5 | static Class<?> forName(String name, boolean initialize, ClassLoader loader) 此方法返回的类或接口与给定的字符串名称相关联的Class对象，使用给定的类加载器。 |
| 6 | <A extends Annotation> A getAnnotation(Class<A> annotationClass) 此方法返回这个元素的注解指定类型，如果存在这样的注释，否则返回null。 |
| 7 | Annotation[] getAnnotations() 此方法返回当前这个元素上的所有注释。 |
| 8 | String getCanonicalName() 此方法返回底层类的Java语言规范中定义的标准名称。 |
| 9 | Class<?>[] getClasses() 此方法返回一个包含代表所有的公共类，并且是此Class对象所表示的类的成员接口的Class对象的数组。 |

| | |
|---|---|
| 10 | ClassLoader getClassLoader() 此方法返回类加载器的类。 |
| 11 | Class<?> getComponentType() 此方法返回类表示数组的组件类型。 |
| 12 | Constructor<T> getConstructor(Class<?>... parameterTypes) 该方法返回一个Constructor对象，它反映此Class对象所表示的类的指定公共构造函数。 |
| 13 | Constructor<?>[] getConstructors() 此方法返回一个包含某些Constructor对象反映此Class对象所表示类的所有公共构造一个数组。 |
| 14 | Annotation[] getDeclaredAnnotations() 此方法返回直接存在于此元素上的所有注释。 |
| 15 | Class<?>[] getDeclaredClasses() 此方法返回Class对象反映声明此Class对象所表示类成员的类和接口组成的数组。 |
| 16 | Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes) 该方法返回一个Constructor对象，它反映此Class对象所表示的类或接口的指定构造函数。 |
| 17 | Constructor<?>[] getDeclaredConstructors() 此方法返回Constructor对象的所有Class对象表示类声明的构造函数的数组。 |
| 18 | Field getDeclaredField(String name) 该方法返回一个Field对象，它反映此Class对象所表示的类或接口指定已声明字段。 |
| 19 | Field[] getDeclaredFields() 此方法返回Field对象的所有Class对象表示的类或接口中声明的字段的数组。 |
| 20 | Method getDeclaredMethod(String name, Class<?>... parameterTypes) 该方法返回一个Method对象，它反映此Class对象所表示的类或接口的指定已声明方法。 |
| 21 | Method[] getDeclaredMethods() 此方法返回Method对象的所有Class对象表示的类或接口中声明的方法的数组。 |
| 22 | Class<?> getDeclaringClass() 如果此Class对象所表示的类或接口是另一个类的成员，返回被声明的类的Class对象。 |
| 23 | Class<?> getEnclosingClass() 此方法返回直接封闭类的底层类。 |
| 24 | Constructor<?> getEnclosingConstructor() 如果此Class对象表示一个构造函数中的一个本地或匿名类，则返回一个代表底层类的立即封闭构造函数构造对象。 |
| 25 | Method getEnclosingMethod() 如果此Class对象表示的方法中的一个本地或匿名类，则返回一个代表底层类的立即封闭方法的Method对象。 |
| 26 | T[] getEnumConstants() 此方法返回枚举类，如果此Class对象不表示枚举类型返回空元素。 |
| 27 | Field getField(String name) 该方法返回一个Field对象，它反映此Class对象所表示的类或接口的指定公共成员字段。 |

| | |
|---|---|
| 28 | Field[] getFields() 此方法返回一个包含Field对象反映此Class对象所表示的类或接口的所有可访问公共字段的数组。 |
| 29 | Type[] getGenericInterfaces() 此方法返回表示由该对象表示的类或接口直接实现的接口类型。 |
| 30 | Type getGenericSuperclass() 此方法返回表示此Class所表示的实体（类，接口，基本类型或void）的直接超类的类型。 |
| 31 | Class<?>[] getInterfaces() 此方法确定由该对象表示的类或接口实现的接口。 |
| 32 | Method getMethod(String name, Class<?>... parameterTypes) 该方法返回一个Method对象，它反映此Class对象所表示的类或接口的指定公共成员方法。 |
| 33 | Method[] getMethods() 此方法返回一个包含对象的方法反映的类或接口的所有公共成员方法此Class对象所表示，包括那些由类或接口以及那些从父类继承声明数组。 |
| 34 | int getModifiers() 此方法返回Java语言修饰符为这个类或者接口，编码为一个整数。 |
| 35 | String getName() 此方法返回此Class对象所表示的实体（类，接口，数组类，基本类型或void）的名字，作为一个字符串。 |
| 36 | Package getPackage() 此方法获取这个类的包。 |
| 37 | ProtectionDomain getProtectionDomain() 此方法返回这个类ProtectionDomain。 |
| 38 | URL getResource(String name) 此方法找到具有给定名称的资源。 |
| 39 | InputStream getResourceAsStream(String name) 此方法找到具有给定名称的资源。 |
| 40 | Object[] getSigners() 此方法得到这个类的签名。 |
| 41 | String getSimpleName() 此方法返回底层类的简单名称在源代码中给出。 |
| 42 | Class<? super T> getSuperclass() 此方法返回的类来表示此Class所表示的实体（类，接口，基本类型或void）的超类。 |
| 43 | TypeVariable<Class<T>>[] getTypeParameters() 此方法返回一个代表由GenericDeclaration对象表示的一般声明，在声明的顺序声明的类型变量TypeVariable对象的数组。 |
| 44 | boolean isAnnotation() 如果此Class对象表示一个注释类型此方法返回true。 |
| 45 | boolean isAnnotationPresent(Class<? extends Annotation> annotationClass) 如果一个注解指定类型是存在于此元素上此方法返回true，否则返回false。 |

| | |
|---|---|
| 46 | boolean isAnonymousClass() 当且仅当底层类是匿名类此方法返回true。 |
| 47 | boolean isArray() 此方法确定该Class对象表示一个数组类。 |
| 48 | boolean isAssignableFrom(Class<?> cls) 此方法判定此Class对象所表示的类或接口可以是一样的，或者说是一个超类或超接口，由指定Class参数所表示的类或接口。 |
| 49 | boolean isEnum() 当且仅当这个类被声明为在源代码中的枚举此方法返回true。 |
| 50 | boolean isInstance(Object obj) 此方法确定指定的对象赋值兼容与此Class所表示的对象。 |
| 51 | boolean isInterface() 此方法判定指定Class对象表示一个接口类型。 |
| 52 | boolean isLocalClass() 当且仅当底层类是局部类此方法返回true。 |
| 53 | boolean isMemberClass() 当且仅当底层类是成员类此方法返回true。 |
| 54 | boolean isPrimitive() 此方法确定指定的Class对象表示一个基本类型。 |
| 55 | boolean isSynthetic() 如果这个类是合成的类此方法返回true;否则返回false。 |
| 56 | T newInstance() 此方法创建此Class对象所表示类的新实例。 |
| 57 | String toString() 此方法的对象转换为字符串。 |

# 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.ClassLoader类 - java.lang

**java.lang.ClassLoader** 类是一个对象，它负责加载类。这个类是一个抽象类。它可用于通过安全管理器，以指示安全域。

## 类声明

以下是**java.lang.ClassLoader**类的声明：

```
public abstract class ClassLoader
  extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected ClassLoader()** 这将创建使用该getSystemClassLoader()方法作为父类加载器返回ClassLoader一个新的类加载器。 |
| 2 | **protected ClassLoader(ClassLoader parent)** 这将创建使用指定的父类加载器委派一个新的类加载器。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void clearAssertionStatus() 此方法设置这个类加载器的默认断言状态设置为false，并放弃与类加载器相关的任何包默认值或类断言状态设置。 |
| 2 | protected Class<?> defineClass(String name, byte[] b, int off, int len) 此方法将字节数组转换为Class类的一个实例。 |
| 3 | protected Class<?> defineClass(String name, byte[] b, int off, int len, ProtectionDomain protectionDomain) 此方法将字节数组转换为Class类的一个实例，一个可选ProtectionDomain |
| 4 | protected Class<?> defineClass(String name, ByteBuffer b, ProtectionDomain protectionDomain) 此方法转换ByteBuffer为Class类的一个实例，并带一个可选的ProtectionDomain。 |
| 5 | protected Package definePackage(String name, String specTitle, String specVersion, String specVendor, String implTitle, String implVersion, String implVendor, URL sealBase) 此方法在这个类加载器 |

|  | 定义了一个包。 |
|---|---|
| 6 | protected Class<?> findClass(String name) 此方法找到的类使用指定的二进制名称。 |
| 7 | protected String findLibrary(String libname) 此方法返回的本地库的绝对路径名。 |
| 8 | protected Class<?> findLoadedClass(String name) 此方法返回类，如果这部分代码已经记录由Java虚拟机作为一类具有二进制名称的启动加载器给定二进制名称。 |
| 9 | protected URL findResource(String name) 此方法找到具有给定名称的资源。 |
| 10 | protected Enumeration<URL> findResources(String name) 此方法返回表示所有具有给定名称的资源的URL对象的枚举。 |
| 11 | protected Class<?> findSystemClass(String name) 此方法找到的类使用指定的二进制名称，如果有必要加载它。 |
| 12 | protected Package getPackage(String name) 此方法返回一个已经被这个类装载器，或任何其祖先定义的包。 |
| 13 | protected Package[] getPackages() 此方法返回所有由这个类装载器和它的祖先定义的包。 |
| 14 | ClassLoader getParent() 此方法返回父类加载器委派。 |
| 15 | URL getResource(String name) 此方法找到具有给定名称的资源。 |
| 16 | InputStream getResourceAsStream(String name) 该方法用于读出指定的资源返回一个输入流。 |
| 17 | Enumeration<URL> getResources(String name) 此方法找到的所有资源与给定的名字。 |
| 18 | static ClassLoader getSystemClassLoader() 此方法返回系统类加载器委派。 |
| 19 | static URL getSystemResource(String name) 此方法找到的用来加载类的搜索路径指定名称的资源。 |
| 20 | static InputStream getSystemResourceAsStream(String name) 此方法是打开用于读取，从用来加载类的搜索路径指定名称的资源。 |
| 21 | static Enumeration<URL> getSystemResources(String name) 此方法找到的用来加载类的搜索路径指定名称的所有资源。 |
| 22 | Class<?> loadClass(String name) 此方法加载类指定二进制名称。 |
| 23 | protected Class<?> loadClass(String name, boolean resolve) 此方法加载类指定二进制名称。 |
| 24 | protected void resolveClass(Class<?> c) 此方法连接指定的类。 |

| | |
|---|---|
| 25 | void setClassAssertionStatus(String className, boolean enabled) 此方法设置在此类加载器及其包含的命名的顶级阶层和任何嵌套类所需的断言状态。 |
| 26 | void setDefaultAssertionStatus(boolean enabled) 此方法设置这个类加载器的默认断言状态。 |
| 27 | void setPackageAssertionStatus(String packageName, boolean enabled) 此方法设置为指定包的包默认断言状态。 |
| 28 | protected void setSigners(Class<?> c, Object[] signers) 此方法设置一个类的签名。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Compiler类 - java.lang

**java.lang.Compiler** 类提供支持Java到本机代码编译器和相关服务。在设计上，它作为一个占位符在JIT编译器实现。

## 类声明

以下是**java.lang.Compiler**类的声明：

```
public final class Compiler
   extends Object
```

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static Object command(Object any) 此方法检查参数类型及其字段，并执行一些文档操作。 |
| 2 | static boolean compileClass(Class<?> clazz) 此方法编译指定的类。 |
| 3 | static void disable() 此方法使编译器停止运作。 |
| 4 | static boolean compileClasses(String string) 此方法编译，其名称与指定字符串匹配的所有类。 |
| 5 | static void enable() 此方法会导致编译器恢复运行。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Double类 - java.lang

**java.lang.Double** 类包装了一个基本类型double 在对象中的值。 Double类型的对象包含一个字段，它的类型是double。

## 类声明

以下是**java.lang.Double**类的声明：

```
public final class Double
  extends Number
     implements Comparable<Double>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Double(double value)** 此构造一个新分配的Double对象，表示原始double参数。 |
| 2 | **Double(String s)** 此构造一个新分配的Double对象，表示double类型的字符串表示的浮点值。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | byte byteValue() 此方法(通过转换成一个字节)返回此Double为一个字节的值。 |
| 2 | static int compare(double d1, double d2) 此方法比较两个指定的double值。 |
| 3 | int compareTo(Double anotherDouble) 此方法比较两个指定的double值。 |
| 4 | static long doubleToLongBits(double value) 此方法返回根据IEEE754浮点“双精度格式”位布局，返回指定浮点值的表示。 |
| 5 | static long doubleToRawLongBits(double value) 此方法返回根据IEEE754浮点“双精度格式”位布局，不是非数字（NaN）值，返回指定浮点值的表示。 |
| | |

| | |
|---|---|
| 6 | double doubleValue() 此方法返回根据IEEE754浮点“双精度格式”位布局，不是非数字（NaN）值，返回指定浮点值的表示。 |
| 7 | boolean equals(Object obj) 此方法比较这个对象与指定对象。 |
| 8 | float floatValue() 此方法返回当前Double对象的浮点值。 |
| 9 | int hashCode() 此方法返回此Double对象的哈希码。 |
| 10 | int intValue() 此方法（通过转换成int类型）返回此Double为一个int值。 |
| 11 | boolean isInfinite() 如果这个Double 值是无限大此方法返回true，否则返回false。 |
| 12 | static boolean isInfinite(double v) 如果指定的数是无限大此方法返回true，否则返回false。 |
| 13 | boolean isNaN() 如果这Double值不是非数字（NAN）此方法返回true，否则返回false。 |
| 14 | static boolean isNaN(double v) 如果指定的数不是非数字（NAN）的值此方法返回true，否则返回false。 |
| 15 | static double longBitsToDouble(long bits) 此方法返回对应于给定的位表示double值。 |
| 16 | long longValue() 此方法（通过转换成long类型）返回此Double作为long值。 |
| 17 | static double parseDouble(String s) 该方法返回一个新的double初始化为指定字符串表示的值，通过Double类的valueOf方法的执行。 |
| 18 | short shortValue() 此方法（通过转换成short）返回此Double作为short的值。 |
| 19 | static String toHexString(double d) 此方法返回double参数的十六进制字符串表示形式。 |
| 20 | String toString() 此方法返回此Double对象的字符串表示形式。 |
| 21 | static String toString(double d) 此方法返回double参数的字符串表示形式。 |
| 22 | static Double valueOf(double d) 此方法返回一个表示指定的double值的Double实例。 |
| 23 | static Double valueOf(String s) 此方法返回持有参数字符串s表示double值的Double对象。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Enum类 - java.lang

java.lang.Enum 类是所有Java语言枚举类型的公共基类。

## 类声明

以下是java.lang.Enum类的声明：

```
public abstract class Enum<E extends Enum<E>>
   extends Object
      implements Comparable<E>, Serializable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected Enum(String name, int ordinal)** 这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | protected Object clone() 此方法将抛出CloneNotSupportedException异常。 |
| 2 | int compareTo(E o) 此方法比较此枚举与指定对象的顺序。 |
| 3 | boolean equals(Object other) 如果指定的对象等于此枚举常量此方法返回true。 |
| 4 | protected void finalize() 此方法返回枚举类不能有finalize方法。 |
| 5 | Class<E> getDeclaringClass() 此方法返回对应于此枚举常量的枚举类型的Class对象。 |
| 6 | int hashCode() 此方法返回枚举常量的哈希码。 |
| 7 | String name() 此方法返回枚举常量的名称，正是因为在枚举声明中声明。 |
| 8 | int ordinal() 此方法返回枚举常量的序数(它在枚举声明，其中初始常量分配的零序位)。 |
| 9 | String toString() 此方法返回枚举常量的名称，它包含在声明中。 |
| 10 | static <T extends Enum<T>> T valueOf(Class<T> enumType, String name) 此方法返回具有指定名称的指定枚举类型的枚举常量。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Float类 - java.lang

**java.lang.Float** 类包装float基本类型的对象的值。 Float类型的对象包含一个字段，它的类型为float。

## 类声明

以下是**java.lang.Float**类的声明：

```
public final class Float
   extends Number
      implements Comparable<Float>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Float(double value)** 此构造一个新分配的Float对象，表示转换为float类型的参数。 |
| 2 | **Float(float value)** 此构造一个新分配的Float对象，它表示基本float参数。 |
| 3 | **Float(String s)** 此构造一个新分配的Float对象，表示float类型的字符串表示的浮点值。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | byte byteValue() 此方法(通过转换成一个字节)返回当前Float为一个字节的值。 |
| 2 | static int compare(float f1, float f2) 此方法比较两个指定的float值。 |
| 3 | int compareTo(Float anotherFloat) 此方法比较两个浮点数的对象数值。 |
| 4 | double doubleValue() 此方法返回该Float对象的double值。 |
| 5 | boolean equals(Object obj) 此方法比较与指定对象与这个对象。 |
| 6 | static int floatToIntBits(float value) 此方法返回根据IEEE754浮点“单一格式”位布局，返回指定浮点值的表示。 |
| | |

| | |
|---|---|
| 7 | static int floatToRawIntBits(float value) 此方法返回根据IEEE754浮点"单一格式"位布局，不是非数字（NaN）值，返回表示指定浮点值。 |
| 8 | float floatValue() 此方法返回该Float对象的浮点值。 |
| 9 | int hashCode() 此方法返回这个Float对象的哈希码。 |
| 10 | static float intBitsToFloat(int bits) 此方法返回对应于给定的位表示的float值。 |
| 11 | int intValue() 此方法（通过转换成int类型）返回当前Float对象对应的整型值。 |
| 12 | boolean isInfinite() 如果这个Float值是无限大此方法返回true，否则返回false。 |
| 13 | static boolean isInfinite(float v) 如果指定的数字是无限大此方法返回true，否则返回false。 |
| 14 | boolean isNaN() 如果这个浮点值不是非数字（NAN）此方法返回true，否则返回false。 |
| 15 | static boolean isNaN(float v) 如果指定的数字不是非数字（NAN）值此方法返回true，否则返回false。 |
| 16 | long longValue() 此方法（通过转换成long类型）返回当前Float的long值。 |
| 17 | static float parseFloat(String s) 此方法返回指定String表示的初始化新的float，通过Float类的valueOf方法执行结果的值。 |
| 18 | short shortValue() 此方法（通过转换成short）返回此Float的short值。 |
| 19 | static String toHexString(float f) 此方法返回float参数的十六进制字符串表示形式。 |
| 20 | String toString() 此方法返回该Float对象的字符串表示形式。 |
| 21 | static String toString(float f) 此方法返回float参数的字符串表示形式 |
| 22 | static Float valueOf(float f) 此方法返回一个Float实例表示指定浮点值。 |
| 23 | static Float valueOf(String s) 该方法返回一个Float对象持有参数字符串s表示的float值。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.InheritableThreadLocal类 - java.lang

**java.lang.InheritableThreadLocal** 类扩展了ThreadLocal从父线程提供的值来继承子线程：创建一个子线程的时候，子的所有可继承线程局部变量可以是其父有接收值的初始值。

## 类声明

以下是**java.lang.InheritableThreadLocal**类的声明：

```
public class InheritableThreadLocal<T>
   extends ThreadLocal<T>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **public class InheritableThreadLocal<T>extends ThreadLocal<T>**<br>这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | protected T childValue(T parentValue) 此方法计算，子类在这个可继承线程局部变量的父级的值在创建子线程时函数的初始值。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.ThreadLocal
- java.lang.Object

# java.lang.Integer类 - java.lang

java.lang.Integer 类封装了基本类型int的值在一个对象。 Integer类型的对象包含一个字段，它的类型是int。

## 类声明

以下是声明了java.lang.Integer类：

```
public final class Integer
  extends Number
    implements Comparable<Integer>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Integer(int value)** 此构造一个新分配的Integer对象，它表示指定的int值。 |
| 2 | **Integer(String s)** 此构造一个新分配的Integer对象，它表示String参数指定int值。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static int bitCount(int i) 此方法返回一个位在指定的int值的二进制补码表示的数。 |
| 2 | byte byteValue() 此方法返回这个整数作为一个字节的值。 |
| 3 | int compareTo(Integer anotherInteger) 此方法比较两个整数对象的数值。 |
| 4 | static Integer decode(String nm) 此方法解码字符串转换为整数。 |
| 5 | double doubleValue() 此方法返回这个整数作为double值。 |
| 6 | boolean equals(Object obj) 此方法将此对象与指定对象作比较。 |
| 7 | float floatValue() 此方法返回这个整数作为一个float值。 |
|  | static Integer getInteger(String nm) 此方法确定具有指定名称的系统属 |

| | |
|---|---|
| | 性的整数值。 |
| 9 | static Integer getInteger(String nm, int val) 此方法确定具有指定名称的系统属性的整数值。 |
| 10 | static Integer getInteger(String nm, Integer val) 此方法返回具有指定名称的系统属性的整数值。 |
| 11 | int hashCode() 此方法返回这个整数的哈希码。 |
| 12 | static int highestOneBit(int i) 此方法返回一个位在指定的int值，一个int值至多单个1位，在最高位（“最左”）的位置。 |
| 13 | int intValue() 此方法返回这个整数作为一个int值。 |
| 14 | long longValue() 此方法返回这个整数作为long值。 |
| 15 | static int lowestOneBit(int i) 此方法返回一个位在指定的int值，一个int值至多单个1位，在最低阶（“最右”）的位置。 |
| 16 | static int numberOfLeadingZeros(int i) 此方法返回零位的最高位（“最左侧”）之前的数指定的int值的二进制补码表示1比特。 |
| 17 | static int numberOfTrailingZeros(int i) 此方法返回零位以下的最低阶（“最右”）的数在指定的int值的二进制补码表示一比特。 |
| 18 | static int parseInt(String s) 此方法将字符串参数作为有符号十进制整数。 |
| 19 | static int parseInt(String s, int radix) 此方法解析字符串参数作为第二个参数指定的基数有符号整数。 |
| 20 | static int reverse(int i) 此方法返回通过反转的比特的顺序中指定的int值的二进制补码表示法得到的值。 |
| 21 | static int reverseBytes(int i) 此方法返回通过反转指定int值的二进制补码表示的字节的顺序而获得的值。 |
| 22 | static int rotateLeft(int i, int distance) 此方法返回通过旋转由位的指定数左移指定的int值的二进制补码表示法得到的值。 |
| 23 | static int rotateRight(int i, int distance) 此方法返回由右由位的指定数的旋转指定的int值的二进制补码表示法得到的值。 |
| 24 | short shortValue() 此方法返回这个整数作为short值。 |
| 25 | static int signum(int i) 此方法返回指定的int值的正负号函数。 |
| 26 | static String toBinaryString(int i) 此方法返回一个整数参数作为基数为2的无符号整数的字符串表示形式。 |
| 27 | static String toHexString(int i) 此方法返回一个整数参数作为基数为16的无符号整数的字符串表示形式。 |
| 28 | static String toOctalString(int i) 此方法返回一个整数参数作为基数为8无 |

| | |
|---|---|
| 28 | 符号整数的字符串表示形式。 |
| 29 | String toString() 此方法返回一个代表该整数的值的String对象。 |
| 30 | static String toString(int i) 此方法返回一个表示指定整数的String对象。 |
| 31 | static String toString(int i, int radix) 此方法返回由第二个参数指定的基数，第一个参数的字符串表示形式。 |
| 32 | static Integer valueOf(int i) 该方法返回一个整数实例，表示指定的int值。 |
| 33 | static Integer valueOf(String s) 此方法返回一个Integer对象持有指定字符串的值。 |
| 34 | static Integer valueOf(String s, int radix) 此方法返回一个Integer对象保持从指定的String中提取的值时，由第二个参数给出的基数进行分析。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Long类 - java.lang

**java.lang.Long** 类封装了基本类型long对象的值。 long类型的对象包含单个字段类型为long。

## 类声明

以下是**java.lang.Long**类的声明：

```
public final class Long
  extends Number
    implements Comparable<Long>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Long(long value)** 此构造一个新分配Long对象，表示指定long参数。 |
| 2 | **Long(String s)** 此构造一个新分配的Long对象，表示String参数所指定的long 值。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static int bitCount(long i) 此方法返回一个位在指定long值的二进制补码表示的数。 |
| 2 | byte byteValue() 此方法返回Long对象的字节的值。 |
| 3 | int compareTo(Long anotherLong) 这种方法数值比较两个long的对象。 |
| 4 | static Long decode(String nm) 这种方法解码字符串转换为long。 |
| 5 | double doubleValue() 这个方法返回这个Long值作为double值。 |
| 6 | boolean equals(Object obj) 此方法将此对象与指定对象比较。 |
| 7 | float floatValue() 此方法返回这个Long 值为float值。 |
| 8 | static Long getLong(String nm) 此方法确定具有指定名称的系统属性的long值。 |
| | |

| | |
|---|---|
| 9 | 统属性的long值。 |
| 10 | static Long getLong(String nm, Long val) 此方法返回具有指定名称的系统属性的long值。 |
| 11 | int hashCode() 此方法返回这个long的哈希码。 |
| 12 | static long highestOneBit(long i) 此方法返回一个long值至多单个1位，在最高位（"最左"）的位置的一个位在指定long值。 |
| 13 | int intValue() 此方法返回这个Long 作为一个int值。 |
| 14 | long longValue() 此方法返回这个Long 作为long值。 |
| 15 | static long lowestOneBit(long i) 此方法返回一个long值至多单个1位，在最低阶（"最右"）的位置的一个位在指定long值。 |
| 16 | static int numberOfLeadingZeros(long i) 此方法返回零位的最高位（"最左侧"）之前的数指定long值的二进制补码表示一比特。 |
| 17 | static int numberOfTrailingZeros(long i) 此方法返回零位以下的最低阶（"最右"）的数指定long值的二进制补码表示一比特。 |
| 18 | static long parseLong(String s) 此方法解析为有符号十进制long的字符串参数。 |
| 19 | static long parseLong(String s, int radix) 此方法解析为在第二个参数指定的基数有符号long字符串参数。 |
| 20 | static long reverse(long i) 此方法返回通过反转位的顺序在指定long值的二进制补码表示形式而得到的值。 |
| 21 | static long reverseBytes(long i) 此方法返回通过反转指定long值的二进制补码表示的字节的顺序而获得的值。 |
| 22 | static long rotateLeft(long i, int distance) 此方法返回通过旋转的由位的指定数左移指定的long值的二进制补码表示法得到的值。 |
| 23 | static long rotateRight(long i, int distance) 此方法返回右键按位指定数量的旋转的指定long值的二进制补码表示形式而得到的值。 |
| 24 | short shortValue() 此方法返回这个Long的一个short值。 |
| 25 | static int signum(long i) 此方法返回指定long值的正负号函数。 |
| 26 | static String toBinaryString(long i) 此方法返回long参数作为基数为2的无符号整数的字符串表示形式。 |
| 27 | static String toHexString(long i) 此方法返回long参数作为基数为16无符号整数的字符串表示形式。 |
| 28 | static String toOctalString(long i) 此方法返回long参数作为基数8无符号整数的字符串表示形式。 |
| 29 | String toString() 此方法返回一个代表该Long值的String对象。 |

| | |
|---|---|
| 30 | static String toString(long i) 此方法返回一个表示指定long的String对象。 |
| 31 | static String toString(long i, int radix) 此方法返回由第二个参数指定的基数，第一个参数的字符串表示形式。 |
| 32 | static Long valueOf(long l) 此方法返回一个Long实例，表示指定的long值。 |
| 33 | static Long valueOf(String s) 该方法返回一个Long对象持有指定字符串的值。 |
| 34 | static Long valueOf(String s, int radix) 此方法返回一个Long对象从指定String中提取的值，由第二个参数给出的基数进行分析。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Math类 - java.lang

java.lang.Math 类包含用于执行基本数字运算，如指数，对数，平方根和三角函数的方法。

## 类声明

以下是java.lang.Math类的声明：

```
public final class Math
   extends Object
```

## 字段域

以下是java.lang.Math类的字段：

- static double E -- 这个double值比任何其他值更接近于e，自然对数的底数。
- static double PI -- 这个double值，该值是比任何其他更靠近圆周率，圆其直径的圆周的比率。

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static double abs(double a) 此方法返回double值的绝对值。 |
| 2 | static float abs(float a) 该方法返回一个浮点数值的绝对值。 |
| 3 | static int abs(int a) 此方法返回一个int值的绝对值。 |
| 4 | static long abs(long a) 此方法返回一个long值的绝对值。 |
| 5 | static double acos(double a) 此方法返回一个值的反余弦;返回的角度范围在0.0到pi。 |
| 6 | static double asin(double a) 此方法返回一个值的反正弦;返回的角度范围为-pi/ 2到pi/ 2。 |
| 7 | static double atan(double a) 此方法返回一个值的反正切;返回的角度范围为-pi/ 2到pi/ 2。 |
| 8 | static double atan2(double y, double x) 此方法返回从直角坐标（X，Y）为极坐标 (r, theta)转化率角度2θ。 |

| | |
|---|---|
| 9 | static double cbrt(double a) 此方法返回double值的立方根。 |
| 10 | static double ceil(double a) 此方法返回最小的（最接近负无穷大）double值，该值大于或等于参数，并等于某个整数。 |
| 11 | static double copySign(double magnitude, double sign) 此方法返回第二浮点参数符号及第一个浮点参数。 |
| 12 | static float copySign(float magnitude, float sign) 此方法返回第二浮点参数符号及第一个浮点参数。 |
| 13 | static double cos(double a) 此方法返回一个角度的三角余弦值。 |
| 14 | static double cosh(double x) 此方法返回double值的双曲余弦值。 |
| 15 | static double exp(double a) 此方法返回欧拉数e为底的一个double值的次幂。 |
| 16 | static double expm1(double x) 此方法返回 e<sup style="margin: 0px; padding: 0px; font-size: 13px;">x</sup> -1. |
| 17 | static double floor(double a) 此方法返回最大的（最接近正无穷大）double值，该值小于或等于参数，并等于某个整数。 |
| 18 | static int getExponent(double d) 此方法返回double代表使用的无偏指数。 |
| 19 | static int getExponent(float f) 该方法返回一个浮点数的表示所使用的无偏指数。 |
| 20 | static double hypot(double x, double y) 此方法返回 sqrt(x<sup style="margin: 0px; padding: 0px; font-size: 13px;">2</sup> +y<sup style="margin: 0px; padding: 0px; font-size: 13px;">2</sup>) 没有中间溢或下溢。 |
| 21 | static double IEEEremainder(double f1, double f2) 此方法计算的规定的IEEE 754标准的两个参数的余数运算。 |
| 22 | static double log(double a) 此方法返回double值的自然对数（以e为底）。 |
| 23 | static double log10(double a) 此方法返回以10为底的对数的double值。 |
| 24 | static double log1p(double x) 此方法返回参数与1之和的自然对数。 |
| 25 | static double max(double a, double b) 此方法返回两个double值之中的较大值。 |
| 26 | static float max(float a, float b) 此方法返回两个float值的较大那个值。 |
| 27 | static int max(int a, int b) 此方法返回两个int值的较大值。 |
| 28 | static long max(long a, long b) 此方法返回两个long值的较大值。 |
| | |

| | |
|---|---|
| 29 | static double min(double a, double b) 此方法返回两个double的较小值。 |
| 30 | static float min(float a, float b) 此方法返回两个float 的较小值。 |
| 31 | static int min(int a, int b) 此方法返回两个int的较小值。 |
| 32 | static long min(long a, long b) 此方法返回两个long的较小值。 |
| 33 | static double nextAfter(double start, double direction) 此方法返回相邻第二个参数的方向上的第一个参数的浮点数。 |
| 34 | static float nextAfter(float start, double direction) 此方法返回相邻第二个参数的方向上的第一个参数的浮点数。 |
| 35 | static double nextUp(double d) 此方法返回正无穷大的方向靠近d中的浮点值。 |
| 36 | static float nextUp(float f) 此方法返回毗邻f 在正无穷大的方向浮点值。 |
| 37 | static double pow(double a, double b) 此方法返回第一个参数提高到第二个参数的幂值。 |
| 38 | static double random() 该方法返回一个double值，正号并大于或等于0.0并且小于1.0。 |
| 39 | static double rint(double a) 此方法返回参数最接近的double值，并等于某个整数。 |
| 40 | static long round(double a) 此方法返回最接近long的参数。 |
| 41 | static int round(float a) 此方法返回最接近参数的整数。 |
| 42 | static double scalb(double d, int scaleFactor) 此方法返回 d × 2<sup style="margin: 0px; padding: 0px; font-size: 13px;">scaleFactor</sup> 舍入好像由一个单一的执行正确舍入的浮点乘法的double值集合的成员。 |
| 43 | static float scalb(float f, int scaleFactor) 此方法返回 f × 2<sup style="margin: 0px; padding: 0px; font-size: 13px;">scaleFactor</sup> 舍入好像由一个单一的执行正确舍入的浮点乘法的浮点值集合的成员。 |
| 44 | static double signum(double d) 此方法返回参数的符号函数;返回0如果该参数为0；返回1.0如果参数大于0，返回-1.0如果参数小于零。 |
| 45 | static float signum(float f) 此方法返回参数的符号函数;返回0如果该参数为0；返回1.0f如果参数大于0，返回-1.0如果参数小于零。 |
| 46 | static double sin(double a) 这个方法返回double值的双曲正弦值。 |
| 47 | static double sinh(double x) 该方法返回一个double值的双曲正弦值。 |
| 48 | static double sqrt(double a) 此方法返回double值的正确舍入正平方根。 |
| 49 | static double tan(double a) 此方法返回一个角的三角正切值。 |

| | |
|---|---|
| 50 | static double tanh(double x) 此方法返回double值的双曲正切值。 |
| 51 | static double toDegrees(double angrad) 此方法转换以弧度为单位，以度数测量的近似相等的角的角度。 |
| 52 | static double toRadians(double angdeg) 此方法转换为度，以弧度为单位的近似等效角的角度。 |
| 53 | static double ulp(double d) 此方法返回参数的ulp的大小。 |
| 54 | static double ulp(float f) 此方法返回参数的ulp的大小。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Number类 - java.lang

**java.lang.Number** 类是BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short类的超类子类必须提供的方法所代表的数值转换为byte, double, float, int, long, 和 short。

## 类声明

以下是java.lang.Number类的声明：

```
public abstract class Number
   extends Object
      implements Serializable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Number()** 这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | byte byteValue() 此方法返回指定数的字节的值。 |
| 2 | abstract double doubleValue() 此方法返回指定数量的double值。 |
| 3 | abstract float floatValue() 此方法返回指定数为float的值。 |
| 4 | abstract int intValue() 此方法返回指定数字的整型值。 |
| 5 | abstract long longValue() 此方法返回指定数字作为一个long值。 |
| 6 | short shortValue() 此方法返回指定数的short值。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Object类 - java.lang

**java.lang.Object** 类为类层次结构的根。每个类都将对象作为超类。所有对象，包括数组，都实现这个类的方法。

## 类声明

以下是**java.lang.Object**类的声明：

```
public class Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **Object()** 这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | protected Object clone() 此方法创建并返回此对象的一个副本。 |
| 2 | boolean equals(Object obj) 此方法指示某个其他对象是否“等于”这一项。 |
| 3 | protected void finalize() 调用此方法在一个对象在垃圾回收时，垃圾回收器确定不存在对该对象的更多引用。 |
| 4 | Class<?> getClass() 此方法返回运行时此类对象。 |
| 5 | int hashCode() 此方法返回该对象的哈希码值。 |
| 6 | void notify() 此方法唤醒正在等待此对象的监视器上的单个线程。 |
| 7 | void notifyAll() 此方法唤醒正在等待此对象监视器上的所有线程。 |
| 8 | String toString() 此方法返回该对象的字符串表示形式。 |
| 9 | void wait() 此方法导致当前线程等待，直到其他线程调用notify()方法或notifyAll()方法。 |
| 10 | void wait(long timeout) 此方法导致当前线程等待，直到其他线程调用此对象的notify() 方法或notifyAll()方法，或在指定已经过去的时间。 |
| 11 | void wait(long timeout, int nanos) 此方法导致当前线程等待，直到其他线程调用此对象的notify() 方法或notifyAll()方法，或者其他某个线程中断当前线程，或者一定量已过的实时时间。 |

# java.lang.Package类 - java.lang

**java.lang.Package** 类包含有关Java包的实现和规范版本信息

## 类声明

以下是**java.lang.Package**类的声明：

```
public class Package
  extends Object
    implements AnnotatedElement
```

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | <A extends Annotation> A getAnnotation(Class<A> annotationClass) 此方法返回这个元素的注解指定类型，如果这样的注释，否则返回null。 |
| 2 | Annotation[] getAnnotations() 此方法返回当前这个元素上的所有注释。 |
| 3 | Annotation[] getDeclaredAnnotations() 此方法返回直接存在于此元素上的所有注释。 |
| 4 | String getImplementationTitle() 此方法返回这个包的名称 |
| 5 | String getImplementationVendor() 此方法返回提供该实现的组织，供应商或公司的名称。 |
| 6 | String getImplementationVersion() 此方法返回这个实现的版本。 |
| 7 | String getName() 此方法返回这个包的名称。 |
| 8 | static Package getPackage(String name) 此方法通过在调用方的ClassLoader实例名称找到包。 |
| 9 | static Package[] getPackages() 此方法得到所有目前已知的调用方的ClassLoader实例的软件包。 |
| 10 | String getSpecificationTitle() 此方法返回这个包实现该规范的名称 |
| 11 | String getSpecificationVendor() 此方法返回的组织，供应商或公司拥有并维护实现此包的类的规范的名称。 |
| 12 | String getSpecificationVersion() 此方法返回这个包实现该规范的版本号。 |
| 13 | int hashCode() 此方法返回从包名称计算的哈希码。 |
| 14 | boolean isAnnotationPresent(Class<? extends Annotation> annotationClass) 如果一个注解指定类型是存在于此元素上，此方法返回true，否则返回false。 |
| 15 | boolean isCompatibleWith(String desired) 此方法比较这包的规范版本与所需的版本。 |
| 16 | boolean isSealed() 如果这个包是密封此方法返回true。 |
| 17 | boolean isSealed(URL url) 如果这个包是密封的，对于指定的代码源URL此方法返回true。 |
| 18 | String toString() 此方法返回这个包的字符串表示形式。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Process类 - java.lang

java.lang.Process 类提供了从进程进行输入，执行输出到进程，等待进程完成，检查进程的退出状态，并摧毁(杀死)进程的方法。

## 类声明

以下是java.lang.Process类的声明：

```
public abstract class Process
   extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Process()** 这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | abstract void destroy() 这种方法杀死子进程。 |
| 2 | abstract int exitValue() 此方法返回子进程的退出值。 |
| 3 | abstract InputStream getErrorStream() 此方法获取子进程的错误流。 |
| 4 | abstract InputStream getInputStream() 此方法获取子进程的输入流。 |
| 5 | abstract OutputStream getOutputStream() 此方法获取子进程的输出流。 |
| 6 | abstract int waitFor() 此方法导致当前线程等待，如果有必要，直到由该Process对象表示的进程已经终止。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.ProcessBuilder类 - java.lang

**java.lang.ProcessBuilder** 类用于创建操作系统进程。此类是不同步的。

## 类声明

以下是**java.lang.ProcessBuilder**类的声明：

```
public final class ProcessBuilder
   extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **ProcessBuilder(List<String> command)** 此构造一个进程生成器指定的操作系统程序和参数。 |
| 2 | **ProcessBuilder(String... command)** 此构造一个进程生成器指定的操作系统程序和参数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | List<String> command() 此方法返回此进程生成器的操作系统程序和参数。 |
| 2 | ProcessBuilder command(List<String> command) 此方法设置此进程生成器的操作系统程序和参数。 |
| 3 | ProcessBuilder command(String... command) 此方法设置此进程生成器的操作系统程序和参数。 |
| 4 | File directory() 此方法返回此进程生成器的工作目录。 |
| 5 | ProcessBuilder directory(File directory) 此方法设置此进程生成器的工作目录。 |
| 6 | Map<String,String> environment() 此方法返回此进程生成器环境的字符串映射视图。 |
| 7 | boolean redirectErrorStream() 这个方法告诉此进程生成器是否合并标准错误和标准输出。 |
| 8 | ProcessBuilder redirectErrorStream(boolean redirectErrorStream) 此方法设置此进程生成器redirectErrorStream的属性。 |
| 9 | Process start() 这种方法使用此进程生成器的属性一个新的进程。 |

# 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Runtime类 - java.lang

java.lang.Runtime 以下是java.lang.Runtime类的声明：

## 类声明

以下是java.lang.Runtime类的声明：

```
public class Runtime
   extends Object
```

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void addShutdownHook(Thread hook) 此方法注册一个新的虚拟机关闭挂钩。 |
| 2 | int availableProcessors() 此方法返回可用处理器的Java虚拟机的数量。 |
| 3 | Process exec(String command) 此方法在一个单独的进程中执行指定的字符串命令。 |
| 4 | Process exec(String[] cmdarray) 此方法在一个单独的进程中执行指定的命令和参数。 |
| 5 | Process exec(String[] cmdarray, String[] envp) 此方法指定环境独立进程中执行指定的命令和参数。 |
| 6 | Process exec(String[] cmdarray, String[] envp, File dir) 此方法指定环境和工作目录的独立进程中执行指定的命令和参数。 |
| 7 | Process exec(String command, String[] envp) 此方法在指定环境的独立进程中执行指定的字符串命令。 |
| 8 | Process exec(String command, String[] envp, File dir) 此方法在指定环境和工作目录的独立进程中执行指定的字符串命令。 |
| 9 | void exit(int status) 此方法通过发起关闭序列，终止当前正在运行的Java虚拟机。 |
| 10 | long freeMemory() 此方法返回可用内存在Java虚拟机的数量。 |
| 11 | void gc() 这种方法运行垃圾回收器。 |
|  | static Runtime getRuntime() 此方法返回与当前Java应用程序相关的运 |

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void addShutdownHook(Thread hook) 此方法注册一个新的虚拟机关闭挂钩。 |
| 2 | int availableProcessors() 此方法返回可用处理器的Java虚拟机的数量。 |
| 3 | Process exec(String command) 此方法在一个单独的进程中执行指定的字符串命令。 |
| 4 | Process exec(String[] cmdarray) 此方法在一个单独的进程中执行指定的命令和参数。 |
| 5 | Process exec(String[] cmdarray, String[] envp) 此方法指定环境独立进程中执行指定的命令和参数。 |
| 6 | Process exec(String[] cmdarray, String[] envp, File dir) 此方法指定环境和工作目录的独立进程中执行指定的命令和参数。 |
| 7 | Process exec(String command, String[] envp) 此方法在指定环境的独立进程中执行指定的字符串命令。 |
| 8 | Process exec(String command, String[] envp, File dir) 此方法在指定环境和工作目录的独立进程中执行指定的字符串命令。 |
| 9 | void exit(int status) 此方法通过发起关闭序列，终止当前正在运行的Java虚拟机。 |
| 10 | long freeMemory() 此方法返回可用内存在Java虚拟机的数量。 |
| 11 | void gc() 这种方法运行垃圾回收器。 |
| 12 | static Runtime getRuntime() 此方法返回与当前Java应用程序相关的运行时对象。 |
| 13 | void halt(int status) 此方法强行终止当前正在运行的Java虚拟机。 |
| 14 | void load(String filename) 此方法加载指定的文件名作为一个动态库。 |
| 15 | void loadLibrary(String libname) 此方法加载指定的库名的动态库。 |
| 16 | long maxMemory() 此方法返回内存，Java虚拟机将尝试使用的最大数。 |
| 17 | boolean removeShutdownHook(Thread hook) 此方法去注册一个以前注册的虚拟机关闭挂钩。 |
| 18 | void runFinalization() 这种方法运行最后审定的任何对象最后确定方法。 |
| 19 | long totalMemory() 此方法返回存储器中的Java虚拟机的总量。 |
| 20 | void traceInstructions(boolean on) 此方法允许/禁止跟踪指令。 |
| 21 | void traceMethodCalls(boolean on) 此方法允许/禁止的方法调用跟踪。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.RuntimePermission类 - java.lang

**java.lang.RuntimePermission** 类是运行时的权限。通过RuntimePermission包含一个名称(也称为“目标名称”)，但没有动作列表; 要么有指定权限，也可以不用指定。

## 类声明

以下是**java.lang.RuntimePermission**类的声明：

```
public final class RuntimePermission
   extends BasicPermission
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **RuntimePermission(String name)** 这种构造一个新的RuntimePermission具有指定名称。 |
| 2 | **RuntimePermission(String name, String actions)** 该构造具有指定名称的新RuntimePermission对象. |

## 类方法

这个类继承了下面的类方法：

- java.lang.BasicPermission
- java.lang.Permission
- java.lang.Object

# java.lang.SecurityManager类 - java.lang

java.lang.SecurityManager 类允许应用程序实现安全策略。它允许一个应用程序来确定，执行可能不安全或敏感的操作前，操作是什么，它是否正在试图在安全范围内，允许执行的操作。该应用程序可以允许或禁止该操作。

## 类声明

以下是java.lang.SecurityManager类的声明：

```
public class SecurityManager
   extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **SecurityManager()** 这构造了一个新的安全管理器。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void checkAccept(String host, int port) 此方法将抛出一个SecurityException如果调用线程不允许接受来自指定的主机和端口号的套接字连接。 |
| 2 | void checkAccess(Thread t) 此方法将抛出一个SecurityException如果调用线程不允许修改线程的参数。 |
| 3 | void checkAccess(ThreadGroup g) 此方法将抛出一个SecurityException如果调用线程不允许修改线程组参数。 |
| 4 | void checkAwtEventQueueAccess() 此方法将抛出一个SecurityException如果调用线程不允许访问AWT事件队列。 |
| 5 | void checkConnect(String host, int port) 此方法将抛出一个SecurityException如果调用线程不允许打开套接字连接到指定的主机和端口号。 |
| 6 | void checkConnect(String host, int port, Object context) 此方法将抛出一个SecurityException如果指定的安全上下文不允许打开套接字连接到指定的主机和端口号。 |

| | |
|---|---|
| 7 | void checkCreateClassLoader() 此方法将抛出一个SecurityException如果调用线程不允许创建新的类加载器。 |
| 8 | void checkDelete(String file) 此方法将抛出一个SecurityException如果调用线程不允许删除指定的文件。 |
| 9 | void checkExec(String cmd) 此方法将抛出一个SecurityException如果调用线程不允许创建子进程。 |
| 10 | void checkExit(int status) 此方法将抛出一个SecurityException如果调用线程不允许使Java虚拟机暂停指定的状态代码。 |
| 11 | void checkLink(String lib) 此方法将抛出一个SecurityException如果调用线程不允许动态链接由字符串参数文件指定的库代码。 |
| 12 | void checkListen(int port) 此方法将抛出一个SecurityException如果调用线程不允许等待指定的本地端口号的连接请求。 |
| 13 | void checkMemberAccess(Class<?> clazz, int which) 此方法将抛出一个SecurityException如果调用线程不允许访问的成员。 |
| 14 | void checkMulticast(InetAddress maddr) 此方法将抛出一个SecurityException如果调用线程不允许使用（加入/离开/发送/接收）IP多播。 |
| 15 | void checkPackageAccess(String pkg) 此方法将抛出一个SecurityException如果调用线程不允许访问由参数指定的包。 |
| 16 | void checkPackageDefinition(String pkg) 此方法将抛出一个SecurityException如果调用线程不允许在由参数指定的包中定义类。 |
| 17 | void checkPermission(Permission perm) 此方法将抛出一个SecurityException，如果所请求的访问，由给定权限所指定，不是基于当前生效的安全策略不允许的。 |
| 18 | void checkPermission(Permission perm, Object context) 如果指定的安全上下文被拒绝访问的获准指定的资源，此方法将抛出一个SecurityException。 |
| 19 | void checkPrintJobAccess() 此方法将抛出一个SecurityException如果调用线程不允许初始化打印作业请求。 |
| 20 | void checkPropertiesAccess() 此方法将抛出一个SecurityException如果调用线程不允许访问或修改系统属性。 |
| 21 | void checkPropertyAccess(String key) 此方法将抛出一个SecurityException如果调用线程不允许与指定的键名来访问系统属性。 |
| 22 | void checkRead(FileDescriptor fd) 此方法将抛出一个SecurityException如果调用线程不允许从指定的文件描述符读取。 |
| 23 | void checkRead(String file) 此方法将抛出一个SecurityException如果调用线程不允许读字符串参数指定的文件。 |

| | |
|---|---|
| 24 | void checkRead(String file, Object context) 此方法将抛出一个SecurityException如果指定的安全上下文不允许读取字符串参数指定的文件。 |
| 25 | void checkSecurityAccess(String target) 该方法确定与指定的权限目标名称权限是否应该被授予或拒绝。 |
| 26 | void checkSetFactory() 此方法将抛出一个SecurityException如果调用线程不允许设置由ServerSocket或Socket，或使用URL中的流处理程序工厂的套接字工厂。 |
| 27 | void checkSystemClipboardAccess() 此方法将抛出一个SecurityException如果调用线程不允许访问系统剪贴板。 |
| 28 | boolean checkTopLevelWindow(Object window) 如果调用线程不被信任，弹出的窗口参数指出的顶层窗口，此方法返回false。 |
| 29 | void checkWrite(FileDescriptor fd) 此方法将抛出一个SecurityException如果调用线程不允许写入指定的文件描述符。 |
| 30 | void checkWrite(String file) 此方法将抛出一个SecurityException如果调用线程不允许写字符串参数指定的文件。 |
| 31 | protected Class[] getClassContext() 此方法返回当前执行堆栈类的数组。 |
| 32 | Object getSecurityContext() 此方法创建一个对象来封装当前执行环境。 |
| 33 | ThreadGroup getThreadGroup() 此方法返回线程组的实例在这个被调用的时候被创建的新线程。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Short类 - java.lang

**java.lang.Short** 类包基本类型为short 对象的值。 short类型的对象包含一个字段的类型为short。

## 类声明

以下是**java.lang.Short**类的声明：

```
public final class Short
  extends Number
     implements Comparable<Short>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Short(short value)** 此构造一个新分配的Short对象，表示指定的short值。 |
| 2 | **Short(String s)** 此构造一个新分配的Short 对象，表示String参数指定的short值。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | byte byteValue() 此方法返回这个Short为一个字节的值。 |
| 2 | int compareTo(Short anotherShort) 此方法在数字上比较两个Short对象。 |
| 3 | static Short decode(String nm) 这种方法解码字符串转换为Short。 |
| 4 | double doubleValue() 此方法返回这个Short为double类型的值。 |
| 5 | boolean equals(Object obj) 此方法比较此对象与指定对象。 |
| 6 | float floatValue() 此方法返回这个Short 为浮点数的值。 |
| 7 | int hashCode() 此方法返回这个Short的哈希码。 |
| 8 | int intValue() 此方法返回这个Short的int类型的值。 |
| 9 | long longValue() 此方法返回这个Short为long类型的值。 |
| 10 | static short parseShort(String s) 该方法将字符串参数作为符号的十进制short类型。 |
| 11 | static short parseShort(String s, int radix) 该方法将字符串参数作为有符号short并以第二个参数指定为基数。 |
| 12 | static short reverseBytes(short i) 此方法返回通过反转指定short值的2的补码表示的字节的顺序而获得的值。 |
| 13 | short shortValue() 此方法返回这个Short为short的值。 |
| 14 | String toString() 此方法返回一个代表该Short值的String对象。 |
| 15 | static String toString(short s) 此方法返回一个表示指定short的一个新的String对象。 |
| 16 | static Short valueOf(short s) 此方法返回一个表示指定short值的Short实例。 |
| 17 | static Short valueOf(String s) 此方法返回一个表示指定short值的Short实例。 |
| 18 | static Short valueOf(String s, int radix) 此方法返回保持从指定的String中提取的值时，由第二个参数给出的基数进行分析short的对象。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.StackTraceElement类 - java.lang

**java.lang.StackTraceElement** 类元素代表一个堆栈帧。除了一个在堆栈的顶部所有的栈帧代表一个方法调用。在堆栈顶部的帧表示在将其生成的堆栈跟踪的执行点。

## 类声明

以下是**java.lang.StackTraceElement**类的声明：

```
public final class StackTraceElement
   extends Object
      implements Serializable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **StackTraceElement(String declaringClass, String methodName, String fileName, int lineNumber)** 这将创建一个表示指定执行点的堆栈跟踪元素。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean equals(Object obj) 如果指定对象的实例与该另一个StackTraceElement实例代表相同的执行点，此方法返回true。 |
| 2 | String getClassName() 此方法返回一个包含由该堆栈跟踪元素所表示的执行点类的完全限定名。 |
| 3 | String getFileName() 此方法返回一个包含由该堆栈跟踪元素所表示的执行点的源文件的名称。 |
| 4 | int getLineNumber() 此方法返回一个包含由该堆栈跟踪元素所表示的执行点源行的行号。 |
| 5 | String getMethodName() 此方法返回一个包含由该堆栈跟踪元素所表示的执行点的方法的名称。 |
| 6 | int hashCode() 此方法返回该堆栈跟踪元素的哈希码值。 |
| 7 | boolean isNativeMethod() 如果包含由该堆栈跟踪元素所表示的执行点的方法是一个本地方法，此方法返回true。 |
| 8 | String toString() 此方法返回该堆栈跟踪元素的字符串表示形式。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.StrictMath类 - java.lang

**java.lang.StrictMath** 类包含用于执行基本数字运算，如指数，对数，平方根和三角函数的方法。

## 类声明

以下是**java.lang.StrictMath**类的声明：

```
public final class StrictMath
   extends Object
```

## 字段域

以下是java.lang.StrictMath类的字段：

- static double E -- 这是double值比任何其他值更接近于e，自然对数的底数。
- static double PI -- 这就是double值，该值是比任何其他更靠近圆周率，圆其直径的圆周的比率。

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static double abs(double a) 此方法返回double值的绝对值。 |
| 2 | static float abs(float a) 该方法返回一个浮点数值的绝对值。 |
| 3 | static int abs(int a) 此方法返回一个int值的绝对值。 |
| 4 | static long abs(long a) 此方法返回一个long值的绝对值。 |
| 5 | static double acos(double a) 此方法返回一个值的反余弦;返回的角度范围在0.0到pi。 |
| 6 | static double asin(double a) 此方法返回一个值的反正弦;返回的角度范围为-pi/2到pi/ 2。 |
| 7 | static double atan(double a) 此方法返回一个值的反正切;返回的角度范围为-pi/ 2到pi/ 2。 |
| 8 | static double atan2(double y, double x) 此方法返回从直角坐标 (x, y)为极坐标(r, theta)的转化率角度2θ。 |

| | |
|---|---|
| 9 | static double cbrt(double a) 此方法返回double值的立方根。 |
| 10 | static double ceil(double a) 此方法返回最小的（最接近负无穷大）double值，该值大于或等于参数，并等于某个整数。 |
| 11 | static double copySign(double magnitude, double sign) 此方法返回第一个浮点参数与第二浮点参数符号。 |
| 12 | static float copySign(float magnitude, float sign) 此方法返回第一个浮点参数与第二浮点参数符号。 |
| 13 | static double cos(double a) 此方法返回一个角的三角余弦值。 |
| 14 | static double cosh(double x) 这个方法返回double值的双曲余弦值。 |
| 15 | static double exp(double a) 此方法返回欧拉数e为底的双精度值的幂。 |
| 16 | static double expm1(double x) 这个方法返回 e<sup style="margin: 0px; padding: 0px; font-size: 13px;">x</sup> -1. |
| 17 | static double floor(double a) 此方法返回最大的（最接近正无穷大）double值，该值小于或等于参数，并等于某个整数。 |
| 18 | static int getExponent(double d) 此方法返回double代表使用的无偏指数。 |
| 19 | static int getExponent(float f) 该方法返回一个浮点数的表示所使用的无偏指数。 |
| 20 | static double hypot(double x, double y) 此方法返回 sqrt(x<sup style="margin: 0px; padding: 0px; font-size: 13px;">2</sup> +y<sup style="margin: 0px; padding: 0px; font-size: 13px;">2</sup>) 没有中间溢或下溢。 |
| 21 | static double IEEEremainder(double f1, double f2) 此方法返回double值的自然对数（以e为底）。 |
| 22 | static double log(double a) 此方法返回double值的自然对数（以e为底）。 |
| 23 | static double log10(double a) 此方法返回以10为底的对数的double值。 |
| 24 | static double log1p(double x) 此方法返回参数与1之和的自然对数。 |
| 25 | static double max(double a, double b) 此方法返回两个double值的最大值。 |
| 26 | static float max(float a, float b) 此方法返回两个float值的最大值。 |
| 27 | static int max(int a, int b) 此方法返回两个int值的最大值。 |
| 28 | static long max(long a, long b) 此方法返回两个long值的最大值。 |
| 29 | static double min(double a, double b) 此方法返回两个double值的最小值。 |

| | |
|---|---|
| 30 | static float min(float a, float b) 此方法返回两个float值的最小值。 |
| 31 | static int min(int a, int b) 此方法返回两个int值的最小值。 |
| 32 | static long min(long a, long b) 此方法返回两个long值的最小值。 |
| 33 | static double nextAfter(double start, double direction) 此方法返回相邻的第二个参数的方向的第一个参数的浮点数。 |
| 34 | static float nextAfter(float start, double direction) 此方法返回相邻第一个参数的第二个参数的方向上的浮点数。 |
| 35 | static double nextUp(double d) 此方法返回正无穷大的方向靠近d的浮点值。 |
| 36 | static float nextUp(float f) 此方法返回毗邻f 在正无穷大的方向浮点值。 |
| 37 | static double pow(double a, double b) 此方法返回第一个参数提高到第二个参数的幂值。 |
| 38 | static double random() 该方法返回一个正号double值，大于或等于0.0并且小于1.0。 |
| 39 | static double rint(double a) 此方法返回最接近参数的double值，并等于某个整数。 |
| 40 | static long round(double a) 此方法返回最接近long参数。 |
| 41 | static int round(float a) 此方法返回最接近参数的整数。 |
| 42 | static double scalb(double d, int scaleFactor) 此方法返回 d × 2<sup style="margin: 0px; padding: 0px; font-size: 13px;">scaleFactor</sup> 舍入好像由一个单一的执行正确舍入的浮点乘法的double 值集合的成员。 |
| 43 | static float scalb(float f, int scaleFactor) 此方法返回 f × 2<sup style="margin: 0px; padding: 0px; font-size: 13px;">scaleFactor</sup> 舍入好像由一个单一的执行正确舍入的浮点乘法的float值集合的成员。 |
| 44 | static double signum(double d) scaleFactor) 此方法返回参数的符号函数;如果该参数为0则返回0，如果参数大于零则返回1.0，如果参数小于零则返回-1.0。 |
| 45 | static float signum(float f) 此方法返回参数的符号函数; 如果该参数为0则返回0，如果参数大于零则返回1.0f，如果参数小于零则返回-1.0f。 |
| 46 | static double sin(double a) 此方法返回一个角度的正弦值。 |
| 47 | static double sinh(double x) 这个方法返回double值的双曲正弦值。 |
| 48 | static double sqrt(double a) 这个方法返回double值的舍入正平方根。 |
| 49 | static double tan(double a) 此方法返回一个ang三角正切值 |
| 50 | static double tanh(double x) 这个方法返回double值的双曲正切值。 |

| | |
|---|---|
| 51 | static double toDegrees(double angrad) 此方法转换以弧度为单位，以度数测量的近似相等的角的角度。 |
| 52 | static double toRadians(double angdeg) 此方法转换为度，以弧度为单位的近似等效角的角度。 |
| 53 | static double ulp(double d) 此方法返回参数的ulp的大小。 |
| 54 | static float ulp(float f) 此方法返回参数的ulp的大小。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.String类 - java.lang

**java.lang.String** 类代表字符串。在Java程序中，如“abc”的所有字符串常量，实现为这个类的实例。字符串是常量，它们的值被创建后不能修改。

## 类声明

以下是java.lang.String类的声明：

```
public final class String
  extends Object
    implements Serializable, Comparable<String>, CharSequence
```

## 字段域

以下是为java.lang.String类的字段：

- static Comparator<String> CASE_INSENSITIVE_ORDER -- 这是一个由compareToIgnoreCase比较器的顺序字符串对象。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **String()** 这将初始化一个新创建的String对象，它表示一个空字符序列。 |
| 2 | **String(byte[] bytes)** 这通过使用平台的默认字符集指定的字节数组解码构造一个新的String。 |
| 3 | **String(byte[] bytes, Charset charset)** 这通过使用指定的字符集解码指定的字节数组构造一个新的String。 |
| 4 | **String(byte[] bytes, int offset, int length)** 这通过使用平台的默认字符集的字节的指定子数组解码构造一个新的String |
| 5 | **String(byte[] bytes, int offset, int length, Charset charset)** 这通过使用指定的字符集指定的字节子数组解码构造一个新的String。 |
| 6 | **String(byte[] bytes, int offset, int length, String charsetName)** 这通过使用指定的字符集指定的字节子数组解码构造一个新的String。 |
| 7 | **String(byte[] bytes, String charsetName)** 这通过使用指定的字符集解码指定的字节数组构造一个新的String。 |
| 8 | **String(char[] value)** 这种分配一个新的String，它表示当前包含在字符数组参数的字符序列。 |
| 9 | **String(char[] value, int offset, int count)** 此方法分配一个新的String，它包含字符的字符数组参数的子数组。 |
| 10 | **String(int[] codePoints, int offset, int count)** 此方法分配一个新的String，它包含字符的Unicode代码点数组参数的子数组。 |
| 11 | **String(String original)** 这将初始化一个新创建的String对象，它代表字符作为参数相同的序列;换句话说，新创建的字符串是该参数字符串的一个副本。 |
| 12 | **String(StringBuffer buffer)** 此分配一个包含字符当前包含在字符串缓冲区参数的顺序一个新的字符串。 |
| 13 | **String(StringBuilder builder)** 此分配一个包含字符当前包含在字符串生成器参数的顺序一个新的字符串。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | char charAt(int index) 此方法返回指定索引处的char值。 |
| 2 | int codePointAt(int index) 此方法返回指定索引处的字符(Unicode代码点)。 |
|  | int codePointBefore(int index) 此方法在指定索引之前返回的字符 |

| | |
|---|---|
| | (Unicode代码点)。 |
| 4 | int codePointCount(int beginIndex, int endIndex) 此方法在该字符串的指定文本范围返回Unicode代码点的数。 |
| 5 | int compareTo(String anotherString) 此方法按字典顺序比较两个字符串。 |
| 6 | int compareToIgnoreCase(String str) 此方法按字典顺序比较两个字符串，不区分大小写的差异。 |
| 7 | String concat(String str) 此方法串连指定字符串到该字符串的结尾。 |
| 8 | boolean contains(CharSequence s) 当且仅当此指定序列字符串包含char值此方法返回true。 |
| 9 | boolean contentEquals(CharSequence cs) 此方法比较该字符串与指定的CharSequence。 |
| 10 | boolean contentEquals(StringBuffer sb) 此方法比较该字符串与指定的StringBuffer。 |
| 11 | static String copyValueOf(char[] data) 此方法返回一个表示指定的数组字符序列的字符串。 |
| 12 | static String copyValueOf(char[] data, int offset, int count) 此方法返回一个表示指定的数组中的字符序列的字符串。 |
| 13 | boolean endsWith(String suffix) 此方法测试此字符串是否以指定的后缀结束。 |
| 14 | boolean equals(Object anObject) 此方法让该字符串与指定的对象进行比较。 |
| 15 | boolean equalsIgnoreCase(String anotherString) 此方法让这个字符串与另一个字符串进行比较，忽略大小写的考虑。 |
| 16 | static String format(Locale l, String format, Object... args) 此方法返回使用指定的语言环境，格式字符串和参数将格式化字符串。 |
| 17 | static String format(String format, Object... args) 此方法返回使用指定格式字符串和参数的格式化字符串。 |
| 18 | byte[] getBytes() 此方法将此String解码使用平台的默认字符集的字节序列，并将结果存储到一个新的字节数组。 |
| 19 | byte[] getBytes(Charset charset) 此方法将此String解码使用给定的字符集的字节序列，并将结果存储到一个新的字节数组。 |
| 20 | byte[] getBytes(String charsetName) 此方法将此String解码使用的字符集命名为字节序列，并将结果存储到一个新的字节数组。 |
| 21 | void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) 此方法从此字符串复制的字符到目标字符数组。 |

| | |
|---|---|
| 22 | int hashCode() 此方法返回该字符串的哈希码。 |
| 23 | int indexOf(int ch) 此方法返回该字符串指定字符第一次出现处的索引。 |
| 24 | int indexOf(int ch, int fromIndex) 此方法返回的索引此字符串指定字符第一次出现处，指定开始搜索的索引处。 |
| 25 | int indexOf(String str) 此方法返回该字符串指定的子串中第一次出现处的索引。 |
| 26 | int indexOf(String str, int fromIndex) 此方法返回从指定的索引处索引此字符串指定的子字符串的第一次出现位置。 |
| 27 | String intern() 此方法返回的字符串对象的规范化表示。 |
| 28 | boolean isEmpty() 此方法返回true，当且仅当length()为0。 |
| 29 | int lastIndexOf(int ch) 此方法返回该字符串指定字符最后一次出现处的索引。 |
| 30 | int lastIndexOf(int ch, int fromIndex) 此方法返回的索引此字符串指定字符最后出现处，向后搜索从指定的索引处。 |
| 31 | int lastIndexOf(String str) 此方法返回该字符串指定的子串的最右边出现处的索引。 |
| 32 | int lastIndexOf(String str, int fromIndex) 此方法返回向后搜索从指定的索引位置，此字符串指定的子字符串的最后出现处的索引。 |
| 33 | int length() 这个方法返回这个字符串的长度。 |
| 34 | boolean matches(String regex) 此方法确定这个字符串是否匹配给定正则表达式。 |
| 35 | int offsetByCodePoints(int index, int codePointOffset) 此方法返回这个字符串，是给定codePointOffset码点偏移的索引。 |
| 36 | boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len) 此方法测试如果两个字符串区域是相等忽略大小写。 |
| 37 | boolean regionMatches(int toffset, String other, int ooffset, int len) 此方法测试如果两个字符串区域都是相等。 |
| 38 | String replace(char oldChar, char newChar) 此方法返回通过用newChar更换所有oldChar出现在此字符串，产生一个新的字符串。 |
| 39 | String replace(CharSequence target, CharSequence replacement) 此方法将替换该字符串相匹配的文字目标序列与指定的文字替换序列中的每个子字符串。 |
| 40 | String replaceAll(String regex, String replacement) 此方法将替换此字符串匹配，用给定的正则表达式替换每个子字符串。 |
| 41 | String replaceFirst(String regex, String replacement) 此方法将替换此字 |

| | |
|---|---|
| 42 | String[] split(String regex) 这种分割方法根据给定的正则表达式匹配这个字符串。 |
| 43 | String[] split(String regex, int limit) 此分割方法根据给定的正则表达式匹配这个字符串。 |
| 44 | boolean startsWith(String prefix) 此方法测试此字符串是否在开头使用指定的前缀。 |
| 45 | boolean startsWith(String prefix, int toffset) 此方法测试此字符串开头的指定索引处的子字符串，开始是否使用指定的前缀。 |
| 46 | CharSequence subSequence(int beginIndex, int endIndex) 此方法返回一个新的字符序列，这个序列的子序列。 |
| 47 | String substring(int beginIndex) 此方法返回一个新的字符串，它是此字符串的一个子字符串。 |
| 48 | String substring(int beginIndex, int endIndex) 此方法返回一个新的字符串，它是此字符串的一个子字符串。 |
| 49 | char[] toCharArray() 此方法这个字符串到一个新的字符数组转换。 |
| 50 | String toLowerCase() 此方法将所有在此字符串中的字符使用默认语言环境的规则为小写。 |
| 51 | String toLowerCase(Locale locale) 此方法将所有在此字符串中的字符，以使用给定Locale的规则的小写形式。 |
| 52 | String toString() 此方法返回的字符串本身。 |
| 53 | String toUpperCase() 此方法使用默认语言环境的规则将这个字符串所有的字转为大写。 |
| 54 | String toUpperCase(Locale locale) 此方法将所有在这个字符串的字符使用给定的语言环境的规则转换为大写。 |
| 55 | String trim() 此方法返回字符串的副本，有开头和结尾的空格省略。 |
| 56 | static String valueOf(boolean b) 此方法返回boolean参数的字符串表示形式。 |
| 57 | static String valueOf(char c) 此方法返回char参数的字符串表示形式。 |
| 58 | static String valueOf(char[] data) 此方法返回将char数组参数的字符串表示形式。 |
| 59 | static String valueOf(char[] data, int offset, int count) 此方法返回将char数组参数的特定子数组的字符串表示形式。 |
| 60 | static String valueOf(double d) 此方法返回double参数的字符串表示形式。 |
| 61 | static String valueOf(float f) 此方法返回float参数的字符串表示形式。 |

| | |
|---|---|
| 61 | static String valueOf(float f) 此方法返回float参数的字符串表示形式。 |
| 62 | static String valueOf(int i) 此方法返回int参数的字符串表示形式。 |
| 63 | static String valueOf(long l) 此方法返回long参数的字符串表示形式。 |
| 64 | static String valueOf(Object obj) 此方法返回Object参数的字符串表示形式。 |

# java.lang.StringBuffer类 - java.lang

**java.lang.StringBuffer** 类是字符的线程安全的，可变序列。以下是关于StringBuffer的要点：

- 字符串缓冲区就像是一个字符串，但可以修改。
- 它包含某些特定的字符序列，但该序列的长度和内容可以通过某些方法调用来改变。
- 它们是安全的多线程应用。
- 每个字符串有缓冲区容量。

## 类声明

以下是**java.lang.StringBuffer**类的声明：

```
public final class StringBuffer
   extends Object
      implements Serializable, CharSequence
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **StringBuffer()** 此构造一个不带字符和16个字符初始容量的字符串缓冲区。 |
| 2 | **StringBuffer(CharSequence seq)** 该构造包含指定CharSequence字符串缓冲区的字符相同。 |
| 3 | **StringBuffer(int capacity)** 此构造一个不带字符，指定初始容量的字符串缓冲区。 |
| 4 | **StringBuffer(String str)** 此构造一个字符串缓冲区初始化为指定字符串的内容。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | StringBuffer append(boolean b) 此方法追加布尔boolean 参数到序列的 |

| | |
|---|---|
| 1 | 字符串表示形式 |
| 2 | StringBuffer append(char c) 此方法追加char参数到这一序列的字符串表示形式。 |
| 3 | StringBuffer append(char[] str) 此方法追加char数组参数到这一序列的字符串表示形式。 |
| 4 | StringBuffer append(char[] str, int offset, int len) 此方法追加char数组参数来到此序列的子数组的字符串表示形式。 |
| 5 | StringBuffer append(CharSequence s) 此方法追加指定的CharSequence到这个序列。 |
| 6 | StringBuffer append(CharSequence s, int start, int end) 此方法追加指定的CharSequence到这个序列的子序列。 |
| 7 | StringBuffer append(double d) 此方法追加double参数到序列的字符串表示形式。 |
| 8 | StringBuffer append(float f) 此方法追加float参数到此序列的字符串表示形式。 |
| 9 | StringBuffer append(int i) 此方法追加int参数到此序列的字符串表示形式。 |
| 10 | StringBuffer append(long lng) 此方法的附加参数long到这个序列的字符串表示形式。 |
| 11 | StringBuffer append(Object obj) 此方法追加对象参数的字符串表示形式。 |
| 12 | StringBuffer append(String str) 此方法将指定的字符串追加到此字符序列。 |
| 13 | StringBuffer append(StringBuffer sb) 此方法将指定的StringBuffer追加此序列。 |
| 14 | StringBuffer appendCodePoint(int codePoint) 此方法附加参数代码点到这个序列的字符串表示形式。 |
| 15 | int capacity() 此方法返回当前容量。 |
| 16 | char charAt(int index) 此方法返回此序列中指定索引处的char值。 |
| 17 | int codePointAt(int index) 此方法返回指定索引处的字符（Unicode代码点） |
| 18 | int codePointBefore(int index) 此方法返回指定索引之前的字符（Unicode代码点） |
| 19 | int codePointCount(int beginIndex, int endIndex) 此方法在这个序列中的指定文本范围内返回Unicode代码点数 |
| | |

| | |
|---|---|
| 20 | 的字符。 |
| 21 | StringBuffer deleteCharAt(int index) 此方法删除在该序列指定位置的char |
| 22 | void ensureCapacity(int minimumCapacity) 此方法确保了容量至少等于指定的最小值。 |
| 23 | void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) 此方法从这个序列到目标字符数组dst复制字符。 |
| 24 | int indexOf(String str) 此方法返回该字符串指定的子串中第一次出现处的索引。 |
| 25 | int indexOf(String str, int fromIndex) 此方法返回的索引此字符串指定的子字符串的第一次出现的，从指定的索引处。 |
| 26 | StringBuffer insert(int offset, boolean b) 这种方法插入boolean变量进入这个序列的字符串表示形式。 |
| 27 | StringBuffer insert(int offset, char c) 此方法将char参数插入到这个序列的字符串表示形式。 |
| 28 | StringBuffer insert(int offset, char[] str) 此方法将char数组参数插入到这个序列的字符串表示形式。 |
| 29 | StringBuffer insert(int index, char[] str, int offset, int len) 此方法插入str数组参数此序列的一个子字符串表示形式。 |
| 30 | StringBuffer insert(int dstOffset, CharSequence s) 此方法插入指定的CharSequence到这个序列。 |
| 31 | StringBuffer insert(int dstOffset, CharSequence s, int start, int end) 此方法插入指定的CharSequence子序列到此序列。 |
| 32 | StringBuffer insert(int offset, double d) 此方法插入double参数到这个序列的字符串表示形式。 |
| 33 | StringBuffer insert(int offset, float f) 此方法插入float参数到这个序列的字符串表示形式。 |
| 34 | StringBuffer insert(int offset, int i) 此方法插入第二个int参数到这个序列的字符串表示形式。 |
| 35 | StringBuffer insert(int offset, long l) 此方法插入long参数到这个序列的字符串表示形式。 |
| 36 | StringBuffer insert(int offset, Object obj) 此方法插入对象参数到这个字符序列的字符串表示形式。 |
| 37 | StringBuffer insert(int offset, String str) 此方法插入字符串到这个字符序列。 |
| 38 | int lastIndexOf(String str) 此方法返回该字符串指定的子串的最右边出现 |

| | |
|---|---|
| 38 | 处的索引。 |
| 39 | int lastIndexOf(String str, int fromIndex) 此方法返回该字符串指定的子字符串的最后出现处的索引。 |
| 40 | int length() 此方法返回长度（字符数）。 |
| 41 | int offsetByCodePoints(int index, int codePointOffset) 此方法返回该序列是从给定的索引由codePointOffset代码点偏移中的索引。 |
| 42 | StringBuffer replace(int start, int end, String str) 此方法替换这个序列中的特定字符串的子字符串。 |
| 43 | StringBuffer reverse() 此方法使该字符序列的序列反向替换。 |
| 44 | void setCharAt(int index, char ch) 指定索引处的字符设置为ch。 |
| 45 | void setLength(int newLength) 此方法将字符序列设置长度。 |
| 46 | CharSequence subSequence(int start, int end) 此方法返回一个新的字符序列，为这个序列的子序列。 |
| 47 | String substring(int start) 此方法返回一个新的String，它包含的字符目前包含在此字符序列子序列 |
| 48 | String substring(int start, int end) 此方法返回一个新的String，它包含目前包含在此序列的子序列的字符。 |
| 49 | String toString() 此方法返回表示此序列中数据的字符串。 |
| 50 | void trimToSize() 此方法试图减少用于字符序列的存储。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.StringBuilder类 - java.lang

**java.lang.StringBuilder** 类是字符的可变序列。这提供了一个API和StringBuffer兼容，但不保证同步。

## 类声明

以下是**java.lang.StringBuilder**类的声明：

```
public final class StringBuilder
  extends Object
    implements Serializable, CharSequence
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **StringBuilder()** 此构造一个字符串生成器，在没有字符和16个字符的初始容量。 |
| 2 | **StringBuilder(CharSequence seq)** 此构造一个字符串生成器包含相同字符作为指定的CharSequence。 |
| 3 | **StringBuilder(int capacity)** 此构造一个字符串生成器，在没有字符，由容量参数指定的初始容量。 |
| 4 | **StringBuilder(String str)** 此构造一个字符串生成器初始化为指定字符串的内容。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | StringBuilder append(boolean b) 此方法追加布尔参数到序列的字符串表示形式。 |
| 2 | StringBuilder append(char c) 此方法追加char参数到这一序列的字符串表示形式。 |
| 3 | StringBuilder append(char[] str) This method appends the string representation of the char array argument to this sequence. |
| 4 | StringBuilder append(char[] str, int offset, int len) 此方法追加char数组 |

| | |
|---|---|
| 4 | 参数到此序列的子数组的字符串表示形式。 |
| 5 | StringBuilder append(CharSequence s) 此方法将指定的字符附加到该序列。 |
| 6 | StringBuilder append(CharSequence s, int start, int end) 此方法追加指定的CharSequence追加到这个序列的子序列。 |
| 7 | StringBuilder append(double d) 此方法追加double参数到序列的字符串表示形式。 |
| 8 | StringBuilder append(float f) 此方法追加float参数到序列的字符串表示形式。 |
| 9 | StringBuilder append(int i) 此方法追加int参数到序列的字符串表示形式。 |
| 10 | StringBuilder append(long lng) 此方法追加long参数到序列的字符串表示形式。 |
| 11 | StringBuilder append(Object obj) 此方法追加对象参数到序列的字符串表示形式。 |
| 12 | StringBuilder append(String str) 此方法将指定的字符串追加到此字符序列。 |
| 13 | StringBuilder append(StringBuffer sb) 此方法将指定的StringBuffer追加到此序列。 |
| 14 | StringBuilder appendCodePoint(int codePoint) 此方法附加参数codePoint到这个序列的字符串表示形式。 |
| 15 | int capacity() 此方法返回当前容量。 |
| 16 | char charAt(int index) 此方法返回此序列中指定索引处的char值。 |
| 17 | int codePointAt(int index) 此方法返回指定索引处的字符（Unicode代码点）。 |
| 18 | int codePointBefore(int index) 此方法在指定索引之前返回的字符（Unicode代码点）。 |
| 19 | int codePointCount(int beginIndex, int endIndex) 此方法在这个序列中的指定文本范围内返回Unicode代码点的数量。 |
| 20 | StringBuilder delete(int start, int end) 此方法删除这个序列中的一个子串的字符。 |
| 21 | StringBuilder deleteCharAt(int index) 此方法移除在此序列中的指定位置的字符。 |
| 22 | void ensureCapacity(int minimumCapacity) 此方法确保了容量至少等于指定的最小值。 |

| | |
|---|---|
| 23 | 序列到目标字符数组dst复制的字符。 |
| 24 | int indexOf(String str) 此方法返回该字符串指定的子字符串中第一次出现处的索引。 |
| 25 | int indexOf(String str, int fromIndex) 此方法返回此字符串指定的子字符串的第一次出现的索引，从指定的索引处。 |
| 26 | StringBuilder insert(int offset, boolean b) 这种方法插入boolean变量到这个序列的字符串表示形式。 |
| 27 | StringBuilder insert(int offset, char c) 此方法将char参数插入到这个序列的字符串表示形式。 |
| 28 | StringBuilder insert(int offset, char[] str) 此方法将char数组参数插入到这个序列的字符串表示形式。 |
| 29 | StringBuilder insert(int index, char[] str, int offset, int len) 此方法插入str数组参数到此序列的一个子字符串表示形式。 |
| 30 | StringBuilder insert(int dstOffset, CharSequence s) 此方法插入指定的CharSequence到这个序列。 |
| 31 | StringBuilder insert(int dstOffset, CharSequence s, int start, int end) 此方法插入指定CharSequence的子序列到此序列。 |
| 32 | StringBuilder insert(int offset, double d) 此方法插入double参数到这个序列的字符串表示形式。 |
| 33 | StringBuilder insert(int offset, float f) This method inserts the string representation of the float argument into this sequence. |
| 34 | StringBuilder insert(int offset, int i) 此方法插入第二个int参数到这个序列的字符串表示形式。 |
| 35 | StringBuilder insert(int offset, long l) 此方法插入long参数到这个序列的字符串表示形式。 |
| 36 | StringBuilder insert(int offset, Object obj) 此方法插入的对象参数到这个字符序列的字符串表示形式。 |
| 37 | StringBuilder insert(int offset, String str) 此方法插入字符串到这个字符序列。 |
| 38 | int lastIndexOf(String str) 此方法返回该字符串指定的子串的最右边出现处的索引。 |
| 39 | int lastIndexOf(String str, int fromIndex) 此方法返回该字符串指定的子字符串的最后出现处的索引。 |
| 40 | int length() 此方法返回长度（字符数）。 |
| 41 | int offsetByCodePoints(int index, int codePointOffset) 此方法返回该序列是从给定的索引由codePointOffset代码点偏移中的索引。 |

| | |
|---|---|
| 42 | StringBuilder replace(int start, int end, String str) 此方法在这个序列中的特定字符串中的字符的子字符串替换字符。 |
| 43 | StringBuilder reverse() 此方法使该字符序列的序列的反向替换。 |
| 44 | void setCharAt(int index, char ch) 指定索引处设置为字符ch。 |
| 45 | void setLength(int newLength) 此方法将字符序列的长度。 |
| 46 | CharSequence subSequence(int start, int end) 此方法返回一个新的字符序列，这个序列的子序列。 |
| 47 | String substring(int start) 此方法返回一个新的String，它包含的字符目前包含在此字符序列子序列。 |
| 48 | String substring(int start, int end) 此方法返回一个新的String，它包含的字符目前包含在此序列的子序列。 |
| 49 | String toString() 此方法返回表示此序列中数据的字符串。 |
| 50 | void trimToSize() 此方法试图减少字符序列的存储。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object
- java.lang.CharSequence

# java.lang.System类 - java.lang

**java.lang.System** 类包含一些有用的类字段和方法。它不能被实例化。通过系统提供的工具：

- 标准输出
- 错误输出流
- 标准输入和访问外部定义的属性和环境变量。
- 一种实用程序方法，用于快速复制数组的一部分。
- 加载文件和库文件的方法

## 类声明

以下是**java.lang.System**类的声明：

```
public final class System
   extends Object
```

## 字段域

以下是java.lang.System类的字段：

- static PrintStream err -- 这是“标准”错误输出流。
- static InputStream in -- 这是在“标准”的输入流。
- static PrintStream out -- 这是在“标准”的输出流。

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length) 此方法会复制从指定源数组的数组，开始在指定的位置，到目标数组的指定位置。 |
| 2 | static String clearProperty(String key) 此方法可删除指定键指示的系统属性。 |
| 3 | static Console console() 此方法返回与当前Java虚拟机关联的唯一Console对象（如果有）。 |

| | |
|---|---|
| 4 | static long currentTimeMillis() 此方法返回当前时间，单位为毫秒。 |
| 5 | static void exit(int status) 此方法终止当前正在运行的Java虚拟机。 |
| 6 | static void gc() 此方法运行垃圾回收器。 |
| 7 | static Map<String,String> getenv() 此方法返回当前系统环境的不可改变的字符串映射视图。 |
| 8 | static String getenv(String name) 此方法获取指定的环境变量的值。 |
| 9 | static Properties getProperties() 此方法确定当前系统性能。 |
| 10 | static String getProperty(String key) 此方法获取指定键指示的系统属性。 |
| 11 | static String getProperty(String key, String def) 此方法获取指定键指定的系统属性。 |
| 12 | static SecurityManager getSecurityManager() 此方法得到了系统的安全接口。 |
| 13 | static int identityHashCode(Object x) 此方法返回相同的哈希代码为给定的对象会由默认hashCode方法（返回），给定对象的类是否重写hashCode()。 |
| 14 | static Channel inheritedChannel() 此方法返回从创建此Java虚拟机的实体中继承的通道。 |
| 15 | static void load(String filename) 此方法加载使用从本地文件系统中的指定文件名作为一个动态库中的代码文件。 |
| 16 | static void loadLibrary(String libname) 此方法加载libname指示参数指定的系统库。 |
| 17 | static String mapLibraryName(String libname) 此方法映射库名称为表示本机库的平台特定的字符串。 |
| 18 | static long nanoTime() 此方法返回最精确的可用系统计时器的当前值，以毫微秒。 |
| 19 | static void runFinalization() 此方法运行最后审定的任何对象最后确定方法。 |
| 20 | static void setErr(PrintStream err) 此方法重新分配“标准”错误输出流。 |
| 21 | static void setIn(InputStream in) 此方法重新分配“标准”输入流。 |
| 22 | static void setOut(PrintStream out) 此方法重新分配“标准”输出流。 |
| 23 | static void setProperties(Properties props) 此方法设置系统属性的属性参数。 |
| 24 | static String setProperty(String key, String value) 此方法设置指定键指示的系统属性。 |

| | |
|---|---|
| 25 | static void setSecurityManager(SecurityManager s) 此方法设置系统的安全性。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Thread类 - java.lang

**java.lang.Thread** 类是执行程序中的线程。Java虚拟机允许应用程序具有执行同时运行多个线程。以下是关于主题的要点：

- 每个线程都有一个优先级。线程具有更高的优先级优先执行线程优先级较低
- 每个线程都可以或不可以也被标记为一个守护程序。
- 有两种方法来创建新的执行线程。一种声明一个类Thread的子类
- 另一种方式来创建一个线程是声明实现Runnable接口的类

## 类声明

以下是**java.lang.Thread**类的声明：

```
public class Thread
  extends Object
    implements Runnable
```

## 字段域

以下是java.lang.Thread类的字段：

- static int MAX_PRIORITY -- 这是一个线程可以有最低的优先级。
- static int NORM_PRIORITY -- 这是分配给一个线程的缺省优先级。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Thread()** 这种分配新的Thread对象。 |
| 2 | **Thread(Runnable target)** 这种分配新的Thread对象。 |
| 3 | **Thread(Runnable target, String name)** 这种分配新的Thread对象。 |
| 4 | **Thread(String name)** 这个构造分配新的Thread对象。 |
| 5 | **Thread(ThreadGroup group, Runnable target)** 这种分配新的Thread对象。 |
| 6 | **Thread(ThreadGroup group, Runnable target, String name)** 这种分配新的Thread对象，以便将target作为其运行对象，将指定的name作为其名称，而属于由组所指的线程组。 |
| 7 | **Thread(ThreadGroup group, Runnable target, String name, long stackSize)** 这种分配新的Thread对象，以便将target作为其运行对象，将指定的name作为其名称，所属组所提到的线程组，并具有指定的堆栈大小。 |
| 8 | **Thread(ThreadGroup group, String name)** 这种分配新的Thread对象。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static int activeCount() 此方法返回活动线程的当前线程的线程组中的数量。 |
| 2 | void checkAccess() 此方法确定当前运行的线程是否有权修改该线程。 |
| 3 | protected Object clone() 此方法返回一个克隆，如果此对象的类是可复制的。 |
| 4 | static Thread currentThread() 此方法返回一个引用到当前正在执行的线程对象。 |
| 5 | static void dumpStack() 此方法打印当前线程的堆栈跟踪到标准错误流。 |
| 6 | static int enumerate(Thread[] tarray) 此方法会复制到指定的数组当前线程的线程组及其子组中的每一个活动线程。 |
| 7 | static Map<Thread,StackTraceElement[]> getAllStackTraces() 此方法返回的堆栈跟踪的所有活动线程的映射。 |
| 8 | ClassLoader getContextClassLoader() 此方法返回该线程的上下文类加载器。 |
| | static Thread.UncaughtExceptionHandler |

| | |
|---|---|
| 9 | getDefaultUncaughtExceptionHandler() 此方法返回调用默认的处理程序，当一个线程突然终止由于未捕获到异常。 |
| 10 | long getId() 此方法返回该线程的标识符。 |
| 11 | String getName() 此方法返回该线程的名称。 |
| 12 | int getPriority() 此方法返回该线程的优先级。 |
| 13 | StackTraceElement[] getStackTrace() 此方法返回一个表示该线程的堆栈转储堆栈跟踪元素的数组。 |
| 14 | Thread.State getState() 此方法返回该线程的状态。 |
| 15 | ThreadGroup getThreadGroup() 此方法返回此线程所属的线程组。 |
| 16 | Thread.UncaughtExceptionHandler getUncaughtExceptionHandler() 此方法返回调用的处理时该线程突然终止，由于未捕获到异常。 |
| 17 | static boolean holdsLock(Object obj) 当且仅当当前线程在指定的对象上保持监视器锁此方法返回true。 |
| 18 | void interrupt() 此方法会中断该线程。 |
| 19 | static boolean interrupted() 此方法测试是否当前线程已被中断。 |
| 20 | boolean isAlive() 此方法测试该线程是否活着。 |
| 21 | boolean isDaemon() 此方法测试该线程是否为守护线程。 |
| 22 | boolean isInterrupted() 此方法测试是否该线程已被中断。 |
| 23 | void join() 等待该线程终止。 |
| 24 | void join(long millis) 等待在最长为millis毫秒终止这个线程。 |
| 25 | void join(long millis, int nanos) 等待在最长为millis毫秒+nonos毫秒该线程死亡。 |
| 26 | void run() 如果该线程是使用独立的Runnable运行对象构造的，则该Runnable对象的run方法被调用;否则，此方法不执行任何操作并返回 |
| 27 | void setContextClassLoader(ClassLoader cl) 此方法设置该线程的上下文类加载器。 |
| 28 | void setDaemon(boolean on) 此方法将该线程标记为守护线程或用户线程。 |
| 29 | static void setDefaultUncaughtExceptionHandler(Thread.UncaughtExceptionHandle eh) 此方法设置的默认处理程序时调用用线程突然终止，由于未捕获到异常，并没有其他的处理程序被定义为该线程。 |
| 30 | void setName(String name) 此方法更改为等于参数名该线程的名称。 |
| 31 | void setPriority(int newPriority) This method changes the priority of this |

| | |
|---|---|
| 31 | thread. |
| 32 | void setUncaughtExceptionHandler(Thread.UncaughtExceptionHandler eh) 此方法设置时调用的处理这个线程突然终止由于未捕获到异常。 |
| 33 | static void sleep(long millis) 此方法使当前执行线程休眠（暂停执行）指定的毫秒数，受制于精度和系统计时器和调度程序精度。 |
| 34 | static void sleep(long millis, int nanos) 此方法使当前执行的线程休眠（暂停执行）为指定的毫秒数加纳秒指定数量，受制于精度和系统计时器和调度程序精度。 |
| 35 | void start() 此方法使该线程开始执行; Java虚拟机调用该线程的run方法。 |
| 36 | String toString() 此方法返回该线程的字符串表示形式，包括线程名称，优先级和线程组。 |
| 37 | static void yield() 此方法使当前执行的线程对象来暂停并允许其他线程执行。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.ThreadLocal类 - java.lang

java.lang.ThreadLocal 类提供线程局部变量。

## 类声明

以下是java.lang.ThreadLocal类的声明：

```
public class ThreadLocal<T>
   extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **ThreadLocal()** 创建一个线程局部变量。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | T get() 此方法返回此线程局部变量的当前线程副本中的值。 |
| 2 | protected T initialValue() 此方法返回当前线程的“初始值”此线程局部变量。 |
| 3 | void remove() 此方法删除该线程局部变量的当前线程的值。 |
| 4 | void set(T value) 此方法设置此线程局部变量的当前线程副本中指定的值。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Throwable类 - java.lang

***java.lang.Throwable*** 类是在Java语言中所有错误和异常的超类。只有在这个类(或它的一个子类)的实例对象由Java虚拟机抛出，也可以由Java throw语句抛出。

## 类声明

以下是java.lang.Throwable类的声明：

```
public class Throwable
   extends Object
      implements Serializable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Throwable()** 这种构造一个新的Throwable null作为其详细消息。 |
| 2 | **Throwable(String message)** 构造一个新的Throwable指定详细消息。 |
| 3 | **Throwable(String message, Throwable cause)** 构造一个新的Throwable指定详细消息和原因。 |
| 4 | **Throwable(Throwable cause)** 构造一个新的throwable与指定的原因和详细消息(cause==null ? null : cause.toString())(它通常包含cause的类和详细消息)。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | Throwable fillInStackTrace() 该方法弥补了执行堆栈跟踪。 |
| 2 | Throwable getCause() 此方法返回此的throwable或空的原因， 如果原因不存在或未知。 |
| 3 | String getLocalizedMessage() 此方法创建这个Throwable的本地化描述。 |
| 4 | String getMessage() 此方法返回这个的throwable的详细消息字符串。 |
| 5 | StackTraceElement[] getStackTrace() 此方法提供编程访问由printStackTrace()打印堆栈跟踪信息。 |
| 6 | Throwable initCause(Throwable cause) 这个方法初始化此抛出为指定值的原因。 |
| 7 | void printStackTrace() 此方法打印此抛出其回溯到标准错误流。 |
| 8 | void printStackTrace(PrintStream s) 此方法打印此抛出其回溯到指定的打印流。 |
| 9 | void printStackTrace(PrintWriter s) 此方法打印此抛出其回溯到指定的打印writer。 |
| 10 | void setStackTrace(StackTraceElement[] stackTrace) 此方法设置将返回getStackTrace()由printStackTrace()相关方法打印堆栈跟踪元素。 |
| 11 | String toString() 此方法返回这个的throwable的简单描述。 |

## 方法继承

这个类从以下类继承的方法：

- java.lang.Object

# java.lang.Void类 - java.lang

**java.lang.Void** 类是一个不可实例化的占位符类来保存一个引用代表了Java关键字void的Class对象。

## 类声明

以下是**java.lang.Void**类的声明：

```
public final class Void
   extends Object
```

## 字段域

以下是java.lang.Void类的字段：

- static Class<Void> TYPE -- 此类对象代表伪类型对应于所述关键字void。

## Class Methods

This class inherits methods from the following class:

- java.lang.Object

# java.lang.Errors - java.lang

java.lang.Errors 提供了在Java lang包抛出不同的错误。

## Error 总结

| S.N. | Error & 描述 |
|---|---|
| 1 | **AbstractMethodError** 当应用程序试图调用一个抽象方法此异常引发。 |
| 2 | **AssertionError** 这被抛出的异常表明断言失败。 |
| 3 | **ClassCircularityError** 当一个圆度被检测到，初始化一个类此异常引发。 |
| 4 | **ClassFormatError** 当Java虚拟机试图读取类文件并确定该文件存在格式错误或无法解释为类文件时，此异常引发。 |
| 5 | **Error** 这是一个错误是Throwable的子类，表示严重的问题，合理的应用程序不应该试图捕获。 |
| 6 | **ExceptionInInitializerError** 这是一个意外的异常发生在一个静态初始化的信号。 |
| 7 | **IllegalAccessError** 此被抛出，如果一个应用程序试图访问或修改字段，或调用它不能访问的方法 |
| 8 | **IncompatibleClassChangeError** 这被抛出时，不兼容的类变化已经发生了一些类定义。 |
| 9 | **InstantiationError** 当应用程序试图使用Java的new结构来实例化一个抽象类或接口此异常引发。 |
| 10 | **InternalError** 这是异常指示发生在Java虚拟机的一些意外的内部错误。 |
| 11 | **LinkageError** LinkageError类的子类指示一个类对另一个类的一些依赖。 |
| 12 | **NoClassDefFoundError** 如果Java虚拟机或ClassLoader实例试图在类的定义和类没有定义负载可以发现此异常引发。 |
| 13 | **NoSuchFieldError** 如果一个应用程序试图访问一个对象或修改指定字段此异常引发，并且该对象不再包含该字段。 |
| 14 | **NoSuchMethodError** 如果一个应用程序试图调用指定一个类(静态或实例)的方法，此异常被抛出，而该类已不再具有该方法的定义。 |
| | **OutOfMemoryError** 这被抛出时，Java虚拟机无法分配一个对象，因 |

| | |
|---|---|
| 16 | **StackOverflowError** 当发生堆栈溢出，因为应用程序递归太深而此异常。 |
| 17 | **ThreadDeath** 这是ThreadDeath的一个实例被扔在事主线程时停止方法，Thread类的零参数被调用。 |
| 18 | **UnknownError** 这时候抛出一个未知但严重的异常发生在Java虚拟机。 |
| 19 | **UnsatisfiedLinkError** 这被抛出，如果Java虚拟机无法找到一个方法的适当本地语言定义时声明为native。 |
| 20 | **UnsupportedClassVersionError** 当Java虚拟机试图读取一个类文件，并确定不支持文件中的主要和次要版本号此异常引发。 |
| 21 | **VerifyError** 当“校验器”检测到一个类文件，但良好的，包含着一些内部不一致性或安全性问题，此异常被抛出。 |
| 22 | **VirtualMachineError** 这被抛出，以指示Java虚拟机崩溃或用尽资源仍需要它继续工作。 |

# java.lang.Interfaces - java.lang

java.lang.Interfaces 提供了在Java lang包使用不同的接口。

## Interface Summary

| S.N. | Interface & 描述 |
|---|---|
| 1 | **Appendable** 此是一个对象，其中char序列和值可以附加 |
| 2 | **CharSequence** 这是一个CharSequence为char值的读取顺序。 |
| 3 | **Cloneable** 这是一个类实现了Cloneable接口，以指示Object.clone()方法，它是合法的，该方法使这个类的实例的字段换取字段副本。 |
| 4 | **Comparable<T>** 这就对实现它的每个类的对象进行整体排序。 |
| 5 | **Iterable<T>** 实现此接口允许一个对象是“foreach”语句的目标。 |
| 6 | **Readable** 这是一个可读的字符的来源。 |
| 7 | **Runnable** 这是Runnable接口应该由任何类的实例，旨在通过一个线程执行来实现。 |
| 8 | **Thread.UncaughtExceptionHandler** 这是调用处理程序的接口，当一个线程突然终止由于未捕获到异常。 |

# Java.math 包教程



java.math中包提供用于执行任意精度整数算法（BigInteger）和任意精度小数算法（BigDecimal）。

本参考教程将通过java.math中封装简单实用的方法实例演示使用。

## 读者

参考是为初学者而准备的，帮助他们了解相关的java.math中封装的所有方法的基本功能。

## 必备条件

在开始做练习与使用在此引用给定的各类例子，需要了解基本的Java编程。

# Java.math.BigDecimal类 - Java.math包

**java.math.BigDecimal** 类提供用于算术，刻度操作，舍入，比较，哈希算法和格式转换操作。

toString()方法提供BigDecimal的规范表示。它使用户可以完全控制舍入行为。

提供用于操作BigDecimal规模两种类型的操作：

- 缩放/舍入操作
- 小数点移动操作。

此类及其迭代器实现Comparable接口的所有可选方法。

## 类声明

以下是声明java.math.BigDecimal类：

```
public class BigDecimal
    extends Number
        implements Comparable<BigDecimal>
```

## 字段域

以下是java.math.BigDecimal类中的字段：

- static BigDecimal ONE -- 值为1，使用刻度为0。
- static int ROUND_CEILING -- 舍入模式舍向正无穷。
- static int ROUND_DOWN -- 舍入模式为向零舍入。
- static int ROUND_FLOOR -- 舍入模式接近负无穷大。
- static int ROUND_HALF_DOWN -- 舍入模式舍入到“最近相邻”如果与两个相邻数字的距离相等，在这种情况下，向下取整。
- static int ROUND_HALF_EVEN -- 舍入模式舍对“近邻”如果与两个相邻数字的距离相等，在这种情况下，舍入向着更加相邻。
- static int ROUND_HALF_UP -- 舍入模式舍入到“最近相邻”如果与两个相邻数字的距离相等，在这种情况下范围。
- static int ROUND_UNNECESSARY -- 舍入模式断言请求的操作具有精确的结果，因此不需要舍入。

- static int ROUND_UP -- 舍入模式舍入去零。
- static BigDecimal TEN -- 值为0，使用刻度为0。
- static BigDecimal ZERO -- 值为0，使用刻度为0。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **BigDecimal(BigInteger val)** 这个构造函数是用来将BigInteger转换为BigDecimal。 |
| 2 | **BigDecimal(BigInteger unscaledVal, int scale)** 这个构造函数用于转换为BigInteger非标度值和一个int尺度成一个BigDecimal。 |
| 3 | **BigDecimal(BigInteger unscaledVal, int scale, MathContext mc)** 这个构造函数用于转换为BigInteger非标度值和一个int尺度转换为BigDecimal，有根据上下文设置进行舍入。 |
| 4 | **BigDecimal(BigInteger val, MathContext mc)** 此构造函数用于根据上下文设置将BigInteger转换为BigDecimal舍入。 |
| 5 | **BigDecimal(char[ ] in)** 此构造函数用于将BigDecimal字符数组表示转化为BigDecimal，接受相同的字符序列与BigDecimal（String）构造函数。 |
| 6 | **BigDecimal(char[ ] in, int offset, int len)** 此构造函数用于将BigDecimal的字符数组表示转化为BigDecimal，接受字符与BigDecimal（String）构造方法相同的字符序列，同时允许指定子数组。 |
| 7 | **BigDecimal(char[ ] in, int offset, int len, MathContext mc)** 此构造函数用于将BigDecimal的字符数组表示转化为BigDecimal，接受字符与BigDecimal（String）构造方法相同的字符序列，同时允许指定子数组，并与根据上下文设置进行舍入。 |
| 8 | **BigDecimal(char[ ] in, MathContext mc)** 此构造函数用于将BigDecimal的字符数组表示转化为BigDecimal，接受相同的字符序列与BigDecimal（String）构造和根据上下文设置进行舍入。 |
| 9 | **BigDecimal(double val)** 这个构造函数是用来转换double为一个BigDecimal，它是双的二进制浮点值的精确十进制表示。 |
| 10 | **BigDecimal(double val, MathContext mc)** 这个构造函数是用来转换double为一个BigDecimal，有根据上下文设置进行舍入。 |
| 11 | **BigDecimal(int val)** 这个构造函数是用来转换一个int转换为BigDecimal。 |
| 12 | **BigDecimal(int val, MathContext mc)** 这个构造函数是用来转换一个int转换为BigDecimal，有根据上下文设置进行舍入。 |

| | |
|---|---|
| 13 | **BigDecimal(long val)** 这个构造函数用于转换long为一个BigDecimal。 |
| 14 | **BigDecimal(long val, MathContext mc)** 这个构造函数是将BigInteger转换为BigDecimal。 |
| 15 | **BigDecimal(String val)** 此构造函数用于一个BigDecimal的字符串表示形式转换为BigDecimal。 |
| 16 | **BigDecimal(String val, MathContext mc)** 此构造函数用于将BigDecimal的字符串表示形式转换为BigDecimal，接受相同的字符串作为与BigDecimal（String）构造，并根据上下文设置进行舍入。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | BigDecimal abs() 此方法返回一个BigDecimal，其值是此BigDecimal的绝对值，其标度是this.scale()。 |
| 2 | BigDecimal abs(MathContext mc) 此方法返回一个BigDecimal，其值是此BigDecimal的绝对值，与根据上下文设置进行舍入。 |
| 3 | BigDecimal add(BigDecimal augend) 此方法返回一个BigDecimal，其值为(this + augend)，其标度为max(this.scale(), augend.scale())。 |
| 4 | BigDecimal add(BigDecimal augend, MathContext mc) 此方法返回一个BigDecimal，其值为 (this + augend)，与根据上下文设置进行舍入。 |
| 5 | byte byteValueExact() 这种方法的BigDecimal转换为一个字节，检查丢失的信息。 |
| 6 | int compareTo(BigDecimal val) 这种方法比较BigDecimal与指定的BigDecimal。 |
| 7 | BigDecimal divide(BigDecimal divisor) 此方法返回一个BigDecimal，其值为（this/除数），且其首选标度为（this.scale() - divisor.scale()）;如果准确的商不能表示（因为它有无穷的十进制扩展），则抛出ArithmeticException。 |
| 8 | BigDecimal divide(BigDecimal divisor, int roundingMode) 此方法返回一个BigDecimal，其值为（this/除数），其标度是this.scale()。 |
| 9 | BigDecimal divide(BigDecimal divisor, int scale, int roundingMode) 此方法返回一个BigDecimal，其值为（this/除数），其标度如指定。 |
| 10 | BigDecimal divide(BigDecimal divisor, int scale, RoundingMode roundingMode) 此方法返回一个BigDecimal，其值为（this/除数），其标度为指定。 |
| 11 | BigDecimal divide(BigDecimal divisor, MathContext mc) 此方法返回一个BigDecimal，其值为（this/除数），与根据上下文设置进行舍入。 |

| | |
|---|---|
| 12 | BigDecimal divide(BigDecimal divisor, RoundingMode roundingMode) 此方法返回一个BigDecimal，其值为（this/除数），其标度是this.scale()。 |
| 13 | BigDecimal[ ] divideAndRemainder(BigDecimal divisor) 这个方法返回一个包含divideToIntegralValue结果，其次是剩下的两个操作数的结果的结果由两个元素组成的BigDecimal数组。 |
| 14 | BigDecimal[ ] divideAndRemainder(BigDecimal divisor, MathContext mc) 这个方法返回一个包含divideToIntegralValue的结果，随后其余与上根据上下文设置进行舍入计算两个操作数的结果的结果由两个元素组成的BigDecimal数组。 |
| 15 | BigDecimal divideToIntegralValue(BigDecimal divisor) 此方法返回一个BigDecimal，其值为商（这/除数）的整数部分四舍五入。 |
| 16 | BigDecimal divideToIntegralValue(BigDecimal divisor, MathContext mc) 此方法返回一个BigDecimal，其值是（这/除数）的整数部分。 |
| 17 | double doubleValue() 此方法将BigDecimal转换为double。 |
| 18 | boolean equals(Object x) 这种方法比较BigDecimal与指定对象是否相等。 |
| 19 | float floatValue() 这种方法将BigDecimal转换为float。 |
| 20 | int hashCode() 此方法返回BigDecimal的哈希代码。 |
| 21 | int intValue() 这种方法将BigDecimal转换为int。 |
| 22 | int intValueExact() 这种方法将BigDecimal转换为int，检查丢失的信息。 |
| 23 | long longValue() 这种方法将BigDecimal转换为long。 |
| 24 | long longValueExact() 这种方法将BigDecimal转换为long，检查丢失的信息。 |
| 25 | BigDecimal max(BigDecimal val) 此方法返回此BigDecimal和val的最大值。 |
| 26 | BigDecimal min(BigDecimal val) 此方法返回此BigDecimal和val的最小值。 |
| 27 | BigDecimal movePointLeft(int n) 此方法返回一个BigDecimal，它等效于将该值的小数点移动n位到左边。 |
| 28 | BigDecimal movePointRight(int n) 此方法返回一个BigDecimal，它等效于将该值的小数点移动n位到右边。 |
| 29 | BigDecimal multiply(BigDecimal multiplicand) 此方法返回一个BigDecimal，其值为（this×被乘数），其标度为（this.scale()+multiplicand.scale()）。 |
| | BigDecimal multiply(BigDecimal multiplicand, MathContext mc) 此方法 |

| | |
|---|---|
| 30 | 返回一个BigDecimal，其值为（this×乘数），以根据上下文设置进行舍入。 |
| 31 | BigDecimal negate() 此方法返回一个BigDecimal，其值是（+this），其标度是this.scale()。 |
| 32 | BigDecimal negate(MathContext mc) 此方法返回一个BigDecimal，其值是（-this），根据上下文设置进行舍入。 |
| 33 | BigDecimal plus() 此方法返回一个BigDecimal，其值是（+this），其标度是this.scale()。 |
| 34 | BigDecimal plus(MathContext mc) 此方法返回一个BigDecimal，其值是（+this），根据上下文设置进行舍入。 |
| 35 | BigDecimal pow(int n) 此方法返回一个BigDecimal，其值是(this<sup style="margin: 0px; padding: 0px; font-size: 13px;">n</sup>), 幂被精确计算，使其具有无限精度。 |
| 36 | BigDecimal pow(int n, MathContext mc) 此方法返回一个BigDecimal，其值是 (this<sup style="margin: 0px; padding: 0px; font-size: 13px;">n</sup>). |
| 37 | int precision() 此方法返回此BigDecimal的精度。 |
| 38 | BigDecimal remainder(BigDecimal divisor) 此方法将BigDecimal转换为一个byte，检查丢失的信息。 |
| 39 | BigDecimal remainder(BigDecimal divisor, MathContext mc) 此方法返回一个BigDecimal，其值为（this%除数），根据上下文设置进行舍入。 |
| 40 | BigDecimal round(MathContext mc) 此方法返回根据MathContext设置舍入一个BigDecimal。 |
| 41 | int scale() 此方法返回此BigDecimal的标度。 |
| 42 | BigDecimal scaleByPowerOfTen(int n) 此方法返回一个BigDecimal，其数值等于 (this * 10<sup style="margin: 0px; padding: 0px; font-size: 13px;">n</sup>). |
| 43 | BigDecimal setScale(int newScale) 此方法返回一个BigDecimal，其标度为指定值，其值在数值上等于该BigDecimal。 |
| 44 | BigDecimal setScale(int newScale, int roundingMode) 此方法返回一个BigDecimal，其标度为指定值，其非标度值乘以或除以此BigDecimal的非标度值除以十的次幂，以保持其整体值决定。 |
| 45 | BigDecimal setScale(int newScale, RoundingMode roundingMode) 此方法返回一个BigDecimal，其标度为指定值，其非标度值乘以或除以此BigDecimal的非标度值除以十的次幂，以保持其整体价决定。 |
| 46 | short shortValueExact() 这种方法将BigDecimal转换为short，检查丢失的信息。 |

| | |
|---|---|
| 47 | int signum() 此方法返回此BigDecimal的正负号函数。 |
| 48 | BigDecimal stripTrailingZeros() 此方法返回一个BigDecimal，它在数值上等于这一个，但与从表示形式移除所有尾部零。 |
| 49 | BigDecimal subtract(BigDecimal subtrahend) 此方法返回一个BigDecimal，其值为（this - 减数），其标度为max（this.scale(), subtrahend.scale()）。 |
| 50 | BigDecimal subtract(BigDecimal subtrahend, MathContext mc) 此方法返回一个BigDecimal，其值为（this - 减数），与根据上下文设置进行舍入。 |
| 51 | BigInteger toBigInteger() 这种方法将BigDecimal转换为BigInteger。 |
| 52 | BigInteger toBigIntegerExact() 这种方法将BigDecimal转换为BigInteger，检查丢失的信息。 |
| 53 | String toEngineeringString() 此方法返回此BigDecimal的字符串表示形式，使用工程计数法，如果需要指数。 |
| 54 | String toPlainString() 此方法返回此BigDecimal的字符串表示形式不带指数字段。 |
| 55 | String toString() 此方法返回此BigDecimal的字符串表示形式，用科学记数法，如果需要指数。 |
| 56 | BigDecimal ulp() 此方法返回一个ULP的此BigDecimal的大小，在最后一位的单位。 |
| 57 | BigInteger unscaledValue() 此方法返回一个BigInteger，其值是此BigDecimal的非标度值。 |
| 58 | static BigDecimal valueOf(double val) 这种方法转换double为一个BigDecimal，使用Double.toString（double）方法提供的double的规范化字符串表示形式。 |
| 59 | static BigDecimal valueOf(long val) 这种方法将一个long值转换为BigDecimal带有刻度的零值。 |
| 60 | static BigDecimal valueOf(long unscaledVal, int scale) 这种方法转换long的非标度值和一个int尺度成一个BigDecimal。 |

# Java.math.BigInteger类实例 - Java.math包

**java.math.BigInteger** 类提供操作类似所有Java的基本整数运算符和java.lang.Math中的所有相关的方法。

它还提供了模运算，GCD计算，素性测试，素数生成，位操作，和一些其他杂项业务操作。所有的操作行为，如果BigInteger的二进制补码委托表示法。

算术运算和按位逻辑运算的语义分别类似于那些Java的整数算术运算符和Java的按位整数运算符。移位操作的语义扩展那些Java的移位运算符的允许负移的距离。

比较操作执行有符号整数的比较。提供模块化的算术运算来计算残留，执行幂运算和计算乘法逆。位运算操作对他们的操作数的二进制补码表示的单个位。

在这个类将抛出NullPointerException，在所有方法和构造函数使用时，通过输入任何参数提供一个空的对象引用。

## 类声明

以下是java.math.BigInteger类的声明：

```
public class BigInteger
    extends Number
        implements Comparable<BigInteger>
```

## 字段域

以下是java.math.BigInteger类中的字段：

- static BigInteger ONE -- BigInteger的常量1。
- static BigInteger TEN -- BigInteger的常量10。
- static BigInteger ZERO -- BigInteger的常量0。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **BigInteger(byte[] val)** 这个构造函数用于转换一个字节数组包含BigInteger的二进制补码，以二进制表示成一个BigInteger。 |
| 2 | **BigInteger(int signum, byte[] magnitude)** 此构造函数用于将BigInteger的符号大小表示法转换成一个BigInteger值。 |
| 3 | **BigInteger(int bitLength, int certainty, Random rnd)** 此构造函数用于构造一个随机生成正BigInteger的可能是以指定的bitLength的素数。 |
| 4 | **BigInteger(int numBits, Random rnd)** 此构造函数用于构造一个随机生成的BigInteger，均匀分布在范围0到 (2<sup style="margin: 0px; padding: 0px; font-size: 13px;">numBits</sup> - 1), 包括. |
| 5 | **BigInteger(String val)** 此构造函数用于将BigInteger的十进制字符串表示形式转换成一个BigInteger值。 |
| 6 | **BigInteger(String val, int radix)** 这个构造函数用于转换为BigInteger的指定基数为一个BigInteger的字符串表示形式。 |

# 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | BigInteger abs() 此方法返回一个BigInteger，其值是此BigInteger的绝对值。 |
| 2 | BigInteger add(BigInteger val) 此方法返回一个BigInteger，其值是(this + val). |
| 3 | BigInteger and(BigInteger val) 此方法返回一个BigInteger，其值是 (this & val). |
| 4 | BigInteger andNot(BigInteger val) 此方法返回一个BigInteger，其值是(this & ~val). |
| 5 | int bitCount() 此方法返回此BigInteger的二进制补码表示的位，从符号位不同的数字。 |
| 6 | int bitLength() 此方法返回位在此BigInteger的最小的二进制补码表示的数，不包括符号位。 |
| 7 | BigInteger clearBit(int n) 此方法返回一个BigInteger，其值相当于此BigInteger与指定位清零。 |
| 8 | int compareTo(BigInteger val) 此方法比较此BigInteger与指定的BigInteger。 |
| 9 | BigInteger divide(BigInteger val) 此方法返回一个BigInteger，其值是(this / val). |

| | |
|---|---|
| 10 | BigInteger[ ] divideAndRemainder(BigInteger val) 此方法返回一个包含两个BigIntegers：(this / val) 和 (this % val)，其次是一个数组。 |
| 11 | double doubleValue() 此方法此BigInteger转换为双精度double。 |
| 12 | boolean equals(Object x) 此方法比较此BigInteger与指定对象是否相等。 |
| 13 | BigInteger flipBit(int n) 此方法返回一个BigInteger，其值相当于此BigInteger与指定位翻转。 |
| 14 | float floatValue() 此方法将BigInteger转换为float。 |
| 15 | BigInteger gcd(BigInteger val) 此方法返回一个BigInteger，其值是绝对值的最大公约数：abs(this) 和abs(val)。 |
| 16 | int getLowestSetBit() 此方法返回最右边的（最低阶）的索引在此BigInteger1比特（零比特的数量，以最右侧的1位的右侧）。 |
| 17 | int hashCode() 此方法返回此BigInteger的哈希代码。 |
| 18 | int intValue() 此方法此BigInteger转换为int。 |
| 19 | boolean isProbablePrime(int certainty) 此方法返回true，如果此BigInteger是素数，其绝对复合数则返回false。 |
| 20 | long longValue() 些方法将BigInteger转换为long。 |
| 21 | BigInteger max(BigInteger val) 此方法返回此BigInteger和val的最大值。 |
| 22 | BigInteger min(BigInteger val) 此方法返回此BigInteger和val的最小值。 |
| 23 | BigInteger mod(BigInteger m) 此方法返回一个BigInteger，其值是(this mod m). |
| 24 | BigInteger modInverse(BigInteger m) 此方法返回一个BigInteger，其值是 (this<sup style="margin: 0px; padding: 0px; font-size: 13px;">-1</sup> mod m). |
| 25 | BigInteger modPow(BigInteger exponent, BigInteger m) 此方法返回一个BigInteger，其值是 (this<sup style="margin: 0px; padding: 0px; font-size: 13px;">exponent</sup> mod m). |
| 26 | BigInteger multiply(BigInteger val) 此方法返回一个BigInteger，其值是(this * val). |
| 27 | BigInteger negate() 此方法返回一个BigInteger，其值是 (-this). |
| 28 | BigInteger nextProbablePrime() 此方法返回一个整数大于该BigInteger的可能是素数。 |
| 29 | BigInteger not() 此方法返回一个BigInteger，其值是 (~this). |
| | BigInteger or(BigInteger val) 此方法返回一个BigInteger，其值是 (this \| |

| | val). |
|---|---|
| 31 | BigInteger pow(int exponent) 此方法返回一个BigInteger，其值是 (this<sup style="margin: 0px; padding: 0px; font-size: 13px;">exponent</sup>). |
| 32 | static BigInteger probablePrime(int bitLength, Random rnd) 此方法返回一个正BigInteger的可能是素数，以指定的bitLength。 |
| 33 | BigInteger remainder(BigInteger val) 此方法返回一个BigInteger，其值是 (this % val). |
| 34 | BigInteger setBit(int n) 此方法返回一个BigInteger，其值相当于此BigInteger与指定的位设置。 |
| 35 | BigInteger shiftLeft(int n) 此方法返回一个BigInteger，其值是 (this << n). |
| 36 | BigInteger shiftRight(int n) 此方法返回一个BigInteger，其值是 (this >> n). |
| 37 | int signum() This method returns the signum function of this BigInteger. |
| 38 | BigInteger subtract(BigInteger val) 此方法返回一个BigInteger，其值是 (this - val). |
| 39 | boolean testBit(int n) 此方法返回当且仅当所指定的位被设置为真。 |
| 40 | byte[ ] toByteArray() 此方法返回一个包含此BigInteger的二进制补码表示的字节数组。 |
| 41 | String toString() 此方法返回此BigInteger的十进制字符串表示形式。 |
| 42 | String toString(int radix) 此方法返回在给定的基数以BigInteger的字符串表示形式。 |
| 43 | static BigInteger valueOf(long val) 此方法返回一个BigInteger，其值等于指定long。 |
| 44 | BigInteger xor(BigInteger val) 此方法返回一个BigInteger，其值是 (this ^ val). |

# Java.math.MathContext类实例 - Java.math包

**java.math.MathContext** 类提供了封装上下文设置的不可变对象，并描述数字运算符的某些规则，例如BigDecimal类的实现。

基于独立设置如下：

1. 精度：用于操作的位数;结果四舍五入到这个精度。
2. RoundingMode：一个对象的RoundingMode它指定要用于舍入的算法。

## 类声明

以下是java.math.MathContext类的声明：

```
public final class MathContext
    extends Object
        implements Serializable
```

## 字段

以下是java.math.MathContext类中的字段：

- static MathContext DECIMAL128 -- MathContext对象与精度设置相匹配的是IEEE 754R Decimal128格式，34位数字，并HALF_EVEN，这是IEEE 754R的默认舍入模式。

- static MathContext DECIMAL32 -- MathContext对象与精度设置相匹配的是IEEE 754R Decimal32格式，7位数和HALF_EVEN，这是IEEE 754R的默认舍入模式。

- static MathContext DECIMAL64 -- MathContext对象与精度设置相匹配的是IEEE 754R Decimal64格式，16位数字，并HALF_EVEN，这是IEEE 754R的默认舍入模式。

- static MathContext UNLIMITED -- MathContext对象，其设置有需要的无限精度运算的值。

## 类构造函数

| S.N. | 构造函数与说明 |
| --- | --- |
| 1 | **MathContext(int setPrecision)** 这个构造函数，构造一个新的MathContext与指定的精度和HALF_UP舍入模式。 |
| 2 | **MathContext(int setPrecision, RoundingMode setRoundingMode)** 这个构造函数，构造一个新的MathContext与指定的精度和舍入模式。 |
| 3 | **MathContext(String val)** 这个构造函数，从一个字符串构造一个新的MathContext。 |

# 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | boolean equals(Object x) 此方法比较MathContext与指定对象是否相等。 |
| 2 | int getPrecision() 此方法返回设置的精度。 |
| 3 | RoundingMode getRoundingMode() 此方法返回RoundingMode设置。 |
| 4 | int hashCode() 此方法返回MathContext的哈希代码。 |
| 5 | String toString() 此方法返回当前MathContext的字符串表示形式。 |

# Java.util包教程



java.util包中包含集合框架，collection类，事件模型，日期和时间，国际化和各种实用工具类。

该参考将引导学习java.util包中的实例应用，并提供简单，实用的方法使用示例。

# 读者

该参考为初学者准备，帮助他们了解有关java.util包中可用的所有方法的基本功能。

# 必备条件

在开始做各类在此引用给定的例子，在这里假设读者已经知道了解基本Java编程。

# Java.util.ArrayDeque类 - Java.util包

**java.util.ArrayDeque** 类提供了可调整大小的阵列，并实现了Deque接口。以下是关于阵列双端队列的要点：

- 数组双端队列没有容量限制，使他们增长为必要支持使用。
- 它们不是线程安全的;如果没有外部同步。
- 不支持多线程并发访问。
- null元素被禁止使用在数组deques。
- 它们要比堆栈Stack和LinkedList快。

此类及其迭代器实现Collection和Iteratorinterfaces方法可选。

## 类的声明

以下是java.util.ArrayDeque类的声明：

```
public class ArrayDeque<E>
   extends AbstractCollection<E>
      implements Deque<E>, Cloneable, Serializable
```

这里<E>代表一个元素，它可以是任何类。例如，如果你正在构建一个整数数组列表，那么初始化可为

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **ArrayDeque()** 此构造函数用于创建一个空数组双端队列容纳16个元素的初始容量。 |
| 2 | **ArrayDeque(Collection<? extends E> c)** 此构造函数用于创建一个包含指定集合的元素的双端队列。 |
| 3 | **ArrayDeque(int numElements)** 此构造函数用于创建一个空数组与双端队列的初始容量足以容纳指定的元素数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean add(E e) 此方法将添加指定的元素，在此deque队列的末尾。 |
| 2 | void addFirst(E e) 此方法将添加指定的元素，在此deque队列的前面。 |
| 3 | void addLast(E e) 此方法将插入指定的元素，在此deque队列的末尾。 |
| 4 | void clear() 此方法移除此deque队列的元素。 |
| 5 | ArrayDeque<E> clone() 此方法返回此deque队列的副本。 |
| 6 | boolean contains(Object o) 如果此deque 队列包含指定的元素，此方法返回true。 |
| 7 | Iterator<E> descendingIterator() 此方法返回一个迭代器在此deque队列以逆向顺序的元素。 |
| 8 | E element() 此方法检索，但是不移除此deque队列表示的队列的头部。 |
| 9 | E getFirst() 此方法检索，但是不移除此deque队列的第一个元素。 |
| 10 | E getLast() 此方法检索，但是不移除此deque队列的最后一个元素。 |
| 11 | boolean isEmpty() 如果此deque队列不包含元素，此方法返回true。 |
| 12 | Iterator<E> iterator() 此方法返回一个迭代器在此deque队列的元素。 |
| 13 | boolean offer(E e) 此方法将指定的元素，在此deque队列的末尾。 |
| 14 | boolean offerFirst(E e) 此方法将指定的元素，在此deque队列的前面。 |
| 15 | boolean offerLast(E e) 此方法将指定的元素，在此deque队列的末尾。 |
| 16 | E peek() 此方法检索，但是不移除此deque队列表示的队列的头部，如果此deque队列为空，则返回null。 |
| 17 | E peekFirst() 此方法检索，但是不移除此deque 队列的第一个元素，或者如果此deque 队列为空，则返回null。 |
| 18 | E peekLast() 此方法检索，但是不移除此deque队列的最后一个元素，如果此deque队列为空，则返回null。 |
| 19 | E poll() 此方法检索并移除此deque队列表示的队列的头部，如果此deque队列为空，则返回null。 |
| 20 | E pollFirst() 此方法检索并移除此deque队列的第一个元素，或者如果此deque队列为空，则返回null。 |
| 21 | E pollLast() 此方法检索并移除此deque队列的最后一个元素，如果此deque队列为空，则返回null。 |
| 22 | E pop() 这种方法的此deque队列所表示的堆栈弹出一个元素。 |

| | |
|---|---|
| 23 | void push(E e) 这种方法将元素推入此deque队列所表示的堆栈。 |
| 24 | E remove() 此方法检索并移除此deque队列表示的队列的头部。 |
| 25 | boolean remove(Object o) 此方法从此deque队列中移除指定元素的单个实例。 |
| 26 | E removeFirst() 此方法检索并移除此deque队列的第一个元素。 |
| 27 | boolean removeFirstOccurrence(Object o) 此方法移除此deque队列的指定元素的第一个匹配。 |
| 28 | E removeLast() 此方法检索并移除此deque队列的最后一个元素。 |
| 29 | boolean removeLastOccurrence(Object o) 此方法移除此deque队列的指定元素的最后一次出现。 |
| 30 | int size() 此方法返回在此deque队列的元素个数。 |
| 31 | object[] toArray() 这个方法返回一个包含所有在此deque队列在适当的序列中元素的数组。 |

# Java.util.ArrayList类 - Java.util包

**java.util.ArrayList** 类提供了可调整大小的数组，并实现了List接口。以下是关于ArrayList中的要点：

- 它实现了所有可选的列表操作，并且还允许所有元素，包括空值null。
- 它提供了一些方法来操作内部用来存储列表的数组的大小。
- 相较于LinkedList实现的常数因子较低。

## 类声明

以下是java.util.ArrayList类的声明：

```
public class ArrayList<E>
   extends AbstractList<E>
      implements List<E>, RandomAccess, Cloneable, Serializable
```

这里<E>代表元素。例如，如果正在构建一个整数数组列表，那么初始化为

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **ArrayList()** 此构造函数用于创建一个空的列表，包含10个元素的初始容量。 |
| 2 | **ArrayList(Collection<? extends E> c)** 此构造函数用于创建一个包含指定集合的元素的列表。 |
| 3 | **ArrayList(int initialCapacity)** 此构造函数用于创建一个空列表的初始容量。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean add(E e) 此方法将指定元素追加到此列表的末尾。 |
| | |

| | |
|---|---|
| 2 | void add(int index, E element) 此方法将在此列表中指定位置的指定元素。 |
| 3 | boolean addAll(Collection<? extends E> c) 此方法会将所有指定集合中的元素添加到此列表的结尾，因为它们是由指定collection的迭代器返回的顺序 |
| 4 | boolean addAll(int index, Collection<? extends E> c) 此方法将所有指定集合中的元素插入此列表，在指定的位置开始。 |
| 5 | void clear() 此方法删除所有来自此列表中的元素。 |
| 6 | Object clone() 这个方法返回当前ArrayList实例的浅表副本。 |
| 7 | boolean contains(Object o) 如果此列表包含指定的元素，此方法返回true。 |
| 8 | void ensureCapacity(int minCapacity) 增加此ArrayList的容量。 |
| 9 | E get(int index) 此方法返回的元素在此列表中的指定位置。 |
| 10 | int indexOf(Object o) 此方法返回指定元素的第一个匹配项的索引在此列表中，或者如果此列表中不包含该元素返回-1。 |
| 11 | boolean isEmpty() 如果此列表不包含元素，此方法返回true。 |
| 12 | int lastIndexOf(Object o) 此方法返回指定元素的最后一个匹配项的索引在此列表中，或者-1，如果此列表中不包含该元素。 |
| 13 | E remove(int index) 此方法删除的元素在此列表中的指定位置。 |
| 14 | boolean remove(Object o) 此方法从该列表中首次出现的指定元素，如果它存在。 |
| 15 | protected void removeRange(int fromIndex, int toIndex) 此方法从该列表中删除所有的索引fromIndex（包括）与toIndex（不包括）之间的元素。 |
| 16 | E set(int index, E element) 此方法取代在与指定元素在此列表中指定位置的元素。 |
| 17 | int size() 此方法返回此列表中的元素数。 |
| 18 | Object[] toArray() 此方法返回一个包含所有在此列表中正确的序列中元素的数组（从第一个到最后一个元素）。 |
| 19 | <T> T[] toArray(T[] a) 此方法返回一个包含所有在此列表中正确的序列中的元素（从第一个到最后一个元素）数组;返回数组的运行时类型是指定数组。 |
| 20 | void trimToSize() 此方法修整此ArrayList实例的是列表的当前大小的容量。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.AbstractList
- java.lang.AbstractCollection
- java.util.Object
- java.util.List

# Java.util.Arrays类 - Java.util包

**java.util.Arrays**类包含一个静态的工厂，允许数组被视为列表。以下是关于数组的要点：

- 这个类包含了各种方法来操作数组(比如排序和搜索)。
- 在这个类中的方法抛出NullPointerException如果指定数组引用为null。

## 类声明

以下是java.util.Arrays类的声明：

```
public class Arrays
    extends Object
```

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static <T> List<T> asList(T a) 此方法返回一个受指定数组支持的固定大小的列表。 |
| 2 | static int binarySearch(byte[] a, byte key) 此方法搜索指定的字节数组使用二进制搜索算法来指定值。 |
| 3 | static int binarySearch(byte[] a, int fromIndex, int toIndex, byte key) 此方法搜索范围指定的字节数组使用二进制搜索算法来指定值。 |
| 4 | static int binarySearch(char[] a, char key) 此方法搜索指定字符数组，使用二进制搜索算法来指定值。 |
| 5 | static int binarySearch(char[] a, int fromIndex, int toIndex, char key) 此方法搜索一个范围内的指定字符数组，使用二进制搜索算法来指定值。 |
| 6 | static int binarySearch(double[] a, double key) 此方法搜索指定double数组，使用二进制搜索算法来指定值。 |
| 7 | static int binarySearch(double[] a, int fromIndex, int toIndex, double key) 此方法搜索范围double指定数组，使用二进制搜索算法来指定值。 |
| 8 | static int binarySearch(float[] a, float key) 此方法搜索浮点数的指定数组，使用二进制搜索算法来指定值。 |
| 9 | static int binarySearch(float[] a, int fromIndex, int toIndex, float key) 此方法搜索范围指定浮点数的数组，使用二进制搜索算法来指定值。 |

| | |
|---|---|
| 10 | static int binarySearch(int[] a, int key) 此方法搜索指定的int型数组使用二进制搜索算法来指定值。 |
| 11 | static int binarySearch(int[] a, int fromIndex, int toIndex, int key) 此方法搜索范围指定的int型数组使用二进制搜索算法来指定值。 |
| 12 | static int binarySearch(long[] a, int fromIndex, int toIndex, long key) 此方法搜索范围指定long数组，使用二进制搜索算法来指定值。 |
| 13 | static int binarySearch(long[] a, long key) 此方法搜索指定long数组，使用二进制搜索算法来指定值。 |
| 14 | static int binarySearch(Object[] a, int fromIndex, int toIndex, Object key) 此方法搜索范围指定数组，使用二进制搜索算法来指定对象。 |
| 15 | static int binarySearch(Object[] a, Object key) 此方法搜索指定数组，使用二进制搜索算法来指定对象。 |
| 16 | static int binarySearch(short[] a, int fromIndex, int toIndex, short key) 此方法搜索范围指定short数组，使用二进制搜索算法来指定值。 |
| 17 | static int binarySearch(short[] a, short key) 此方法搜索指定short数组，使用二进制搜索算法来指定值。 |
| 18 | static <T> int binarySearch(T[] a, int fromIndex, int toIndex, T key, Comparator<? super T> c) 此方法搜索范围指定数组，使用二进制搜索算法来指定对象。 |
| 19 | static <T> int binarySearch(T[] a, T key, Comparator<? super T> c) 此方法搜索指定数组，使用二进制搜索算法来指定对象。 |
| 20 | static boolean[] copyOf(boolean[] original, int newLength) 此方法复制指定的数组，截取或填充false（如有必要），以使副本具有指定的长度。 |
| 21 | static byte[] copyOf(byte[] original, int newLength) 此方法复制指定的数组，截取或填充用零（如有必要），以使副本具有指定的长度。 |
| 22 | static char[] copyOf(char[] original, int newLength) 此方法复制指定的数组，截取或填充用null（如有必要），以使副本具有指定的长度。 |
| 23 | static double[] copyOf(double[] original, int newLength) 此方法复制指定的数组，截取或填充用零（如有必要），以使副本具有指定的长度。 |
| 24 | static float[] copyOf(float[] original, int newLength) 此方法复制指定的数组，截取或填充用零（如有必要），以使副本具有指定的长度。 |
| 25 | static int[] copyOf(int[] original, int newLength) 此方法复制指定的数组，截取或填充用零（如有必要），以使副本具有指定的长度。 |
| 26 | static long[] copyOf(long[] original, int newLength) 此方法复制指定的数组，截取或填充用零（如有必要），以使副本具有指定的长度。 |
| | static short[] copyOf(short[] original, int newLength) 此方法复制指定的 |

| | |
|---|---|
| | 数组，截取或填充用零（如有必要），以使副本具有指定的长度。 |
| 28 | static <T> T[] copyOf(T[] original, int newLength) 此方法复制指定的数组，截取或用null填充（如有必要），以使副本具有指定的长度。 |
| 29 | static <T,U> T[] copyOf(U[] original, int newLength, Class<? extends T[]> newType) 此方法复制指定的数组，截取或用null填充（如有必要），以使副本具有指定的长度。 |
| 30 | static boolean[] copyOfRange(boolean[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 31 | static byte[] copyOfRange(byte[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 32 | static char[] copyOfRange(char[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 33 | static double[] copyOfRange(double[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 34 | static float[] copyOfRange(float[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 35 | static int[] copyOfRange(int[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 36 | static long[] copyOfRange(long[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 37 | static short[] copyOfRange(short[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 38 | static <T> T[] copyOfRange(T[] original, int from, int to) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 39 | static <T,U> T[] copyOfRange(U[] original, int from, int to, Class<? extends T[]> newType) 此方法复制指定的数组到一个新的数组的指定范围。 |
| 40 | static boolean deepEquals(Object[] a1, Object[] a2) 如果两个指定数组相等此方法返回true。 |
| 41 | static int deepHashCode(Object[] a) 此方法返回基于指定数组的“深层内容”返回哈希码。 |
| 42 | static String deepToString(Object[] a) 此方法返回指定数组的“深层内容”的字符串表示形式。 |
| 43 | static boolean equals(boolean[] a, boolean[] a2) 如果布尔值的两个指定数组相互相等此方法返回true。 |
| 44 | static boolean equals(byte[] a, byte[] a2) 如果两个指定字节数组相互相等此方法返回true。 |

| | |
|---|---|
| | 等此方法返回true。 |
| 45 | static boolean equals(char[] a, char[] a2) 如果两个指定字符数组相互相等此方法返回true。 |
| 46 | static boolean equals(double[] a, double[] a2) 如果两个指定double数组相互相等此方法返回true。 |
| 47 | static boolean equals(float[] a, float[] a2) 如果浮点数的两个指定数组相互相等此方法返回true。 |
| 48 | static boolean equals(int[] a, int[] a2) 如果整数的两个指定数组相互相等此方法返回true。 |
| 49 | static boolean equals(long[] a, long[] a2) 如果两个指定long数组相互相等此方法返回true。 |
| 50 | static boolean equals(Object[] a, Object[] a2) 如果两个指定对象数组相互相等此方法返回true。 |
| 51 | static boolean equals(short[] a, short[] a2) 如果两个指定对象数组相互相等此方法返回true。 |
| 52 | static void fill(boolean[] a, boolean val) 此方法分配指定指定布尔值数组的每个元素。 |
| 53 | static void fill(boolean[] a, int fromIndex, int toIndex, boolean val) 此方法分配指定指定布尔值数组的指定范围中的每个元素的布尔值。 |
| 54 | static void fill(byte[] a, byte val) 此方法分配指定指定的字节数组的每个元素的字节值。 |
| 55 | static void fill(byte[] a, int fromIndex, int toIndex, byte val) 此方法分配指定的字节数组指定范围中的每个元素的字节值。 |
| 56 | static void fill(char[] a, char val) 此方法分配指定的char值到指定数组的每个元素的字符。 |
| 57 | static void fill(char[] a, int fromIndex, int toIndex, char val) 此方法分配指定的char值的指定数组的指定范围中的每个元素的字符。 |
| 58 | static void fill(double[] a, double val) 此方法分配一个指定的double值到指定数组的每个元素的double值。 |
| 59 | static void fill(double[] a, int fromIndex, int toIndex, double val) 此方法分配一个指定的double值到指定的double数组的指定范围中的每个元素。 |
| 60 | static void fill(float[] a, float val) 此分配方法指定float值数指定数组的每个元素的浮点值。 |
| 61 | static void fill(float[] a, int fromIndex, int toIndex, float val) 此分配方法指定float值数指定数组的指定范围中的每个元素的浮点值。 |
| | static void fill(int[] a, int val) 此分配方法指定数组的指定范围中的每个元 |

| | |
|---|---|
| | 素的int值。 |
| 63 | static void fill(int[] a, int fromIndex, int toIndex, int val) 此分配方法指定数组的指定范围中的每个元素的int值。 |
| 64 | static void fill(long[] a, int fromIndex, int toIndex, long val) 此分配方法指定数组的指定范围中的每个元素的long值。 |
| 65 | static void fill(long[] a, long val) 此分配方法指定long指定数组的每个元素的long值。 |
| 66 | static void fill(Object[] a, int fromIndex, int toIndex, Object val) 此方法分配指定的Object引用的对象的指定数组的指定范围中的每个元素。 |
| 67 | static void fill(Object[] a, Object val) 此方法分配指定的Object引用的对象指定的数组中的每个元素。 |
| 68 | static void fill(short[] a, int fromIndex, int toIndex, short val) 此方法分配指定数组的指定范围中的每个元素的short值。 |
| 69 | static void fill(short[] a, short val) 此方法分配指定short数组的每个元素的short值。 |
| 70 | static int hashCode(boolean[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 71 | static int hashCode(byte[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 72 | static int hashCode(char[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 73 | static int hashCode(double[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 74 | static int hashCode(float[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 75 | static int hashCode(int[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 76 | static int hashCode(long[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 77 | static int hashCode(Object[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 78 | static int hashCode(short[] a) 此方法返回基于指定数组的内容的哈希码。 |
| 79 | static void sort(byte[] a) 此方法指定的字节数组排序按数字升序顺序。 |
| 80 | static void sort(byte[] a, int fromIndex, int toIndex) 此方法对指定范围内指定的字节数组按数字升序顺序。 |
| 81 | static void sort(char[] a) 此方法将char型数组排序按数字升序顺序。 |

| | |
|---|---|
| 81 | static void sort(char[] a) 此方法将char型数组排序按数字升序顺序。 |
| 82 | static void sort(char[] a, int fromIndex, int toIndex) 此方法对指定范围内指定的字符数组按数字升序顺序。 |
| 83 | static void sort(double[] a) 此方法指定double数组排序按数字升序顺序。 |
| 84 | static void sort(double[] a, int fromIndex, int toIndex) 此方法对指定范围的指定double 数组按数字升序顺序。 |
| 85 | static void sort(float[] a) 此方法指定浮点数数组排序按数字升序顺序。 |
| 86 | static void sort(float[] a, int fromIndex, int toIndex) 此方法对指定范围内的指定float数组按数字升序顺序。 |
| 87 | static void sort(int[] a) 此方法指定的int型数组排序按数字升序顺序。 |
| 88 | static void sort(int[] a, int fromIndex, int toIndex) 此方法对指定范围内指定的int型数组按数字升序顺序。 |
| 89 | static void sort(long[] a) 此方法指定的long数组排序按数字升序顺序。 |
| 90 | static void sort(long[] a, int fromIndex, int toIndex) 此方法对指定范围内指定long数组按数字升序顺序。 |
| 91 | static void sort(Object[] a) 根据其元素的自然顺序， 此方法对指定对象数组按升序进行。 |
| 92 | static void sort(Object[] a, int fromIndex, int toIndex) 根据其元素的自然顺序， 此方法对指定对象数组按升序顺序的指定范围。 |
| 93 | static void sort(short[] a) 此方法指定的short 数组排序按数字升序顺序。 |
| 94 | static void sort(short[] a, int fromIndex, int toIndex) 此方法对指定范围内的short指定数组按数字升序顺序。 |
| 95 | static <T> void sort(T[] a, Comparator<? super T> c) 此方法的对象进行排序根据引起的指定比较顺序指定数组。 |
| 96 | static <T> void sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c) 根据引起由指定比较器的顺序此方法排序对象的指定数组的指定范围。 |
| 97 | static String toString(boolean[] a) 此方法返回指定的boolean数组内容的字符串表示形式。 |
| 98 | static String toString(byte[] a) 此方法返回指定的byte数组内容的字符串表示形式。 |
| 99 | static String toString(char[] a) 此方法返回指定的char数组内容的字符串表示形式。 |
| 100 | static String toString(double[] a) 此方法返回指定的double数组内容的字 |

| | |
|---|---|
| 101 | static String toString(float[] a) 此方法返回指定的float数组内容的字符串表示形式。 |
| 102 | static String toString(int[] a) 此方法返回指定的int数组内容的字符串表示形式。 |
| 103 | static String toString(long[] a) 此方法返回指定的long数组内容的字符串表示形式。 |
| 104 | static String toString(Object[] a) 此方法返回指定的对象数组内容的字符串表示形式。 |
| 105 | static String toString(short[] a) 此方法返回指定的short数组内容的字符串表示形式。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# Java.util.BitSet类 - Java.util包

**java.util.BitSet** 类实现位向量作为一个按需增长。以下是关于BitSet中的要点：

- BitSet 是不安全的，除非外部同步多线程使用。
- 集合中的所有位的初始值false。
- 传递一个null参数到BitSet中的任何方法会导致一个NullPointerException。

## 类声明

以下是java.util.BitSet类的声明：

```
public class BitSet
   extends Object
      implements Cloneable, Serializable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **BitSet()** 此构造函数创建一个新的位集。 |
| 2 | **BitSet(int nbits)** 此构造函数创建一个位集，它的初始大小足够大，可以通过0 到 nbits-1范围显式表示索引。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | void and(BitSet set) 此方法执行一个逻辑这个目标位，并设置参数位集合。 |
| 2 | void andNot(BitSet set) 此方法清除所有的位，其相应的位在指定BitSet中设置此BitSet中的位。 |
| 3 | int cardinality() 此方法返回设置为true，此BitSet中的比特数。 |
| 4 | void clear() 此方法将此BitSet中的所有的位设置为false。 |
| 5 | void clear(int bitIndex) 此方法设置由index指定的位为false。 |
| | |

| | |
|---|---|
| | 括）设置位到指定toIndex（不包括）为false。 |
| 7 | Object clone() 此方法克隆该BitSet中，并产生一个新的等于它的BitSet。 |
| 8 | boolean equals(Object obj) 这个方法是比较这个对象与指定对象。 |
| 9 | void flip(int bitIndex) 这种方法在指定索引到它的当前值的补码在设置位。 |
| 10 | void flip(int fromIndex, int toIndex) 此方法设置每个位将指定的fromIndex（包括）到指定的toIndex（不包括）为其当前值的补码。 |
| 11 | boolean get(int bitIndex) 此方法返回具有指定索引的位的值。 |
| 12 | BitSet get(int fromIndex, int toIndex) 此方法返回的位组成一个新BitSet中此BitSet中从fromIndex（包括）到toIndex（不包括）。 |
| 13 | int hashCode() 此方法返回具有指定索引的位的值。 |
| 14 | boolean intersects(BitSet set) 如果指定BitSet中有设置为true，此BitSet中的任何位此方法返回true。 |
| 15 | boolean isEmpty() 如果此BitSet中没有包含位被设置为true，此方法返回true。 |
| 16 | int length() 此方法返回此BitSet的“逻辑大小”：在BitSet中最高设置位加一的索引。 |
| 17 | int nextClearBit(int fromIndex) 此方法返回被设置为出现或之后指定的起始索引false的位的索引。 |
| 18 | int nextSetBit(int fromIndex) 此方法返回被设置为出现或之后指定的起始索引true的位的索引。 |
| 19 | void or(BitSet set) 此方法执行的逻辑该位或设置该位设置参数。 |
| 20 | void set(int bitIndex) 这种方法在指定索引设置该位为true。 |
| 21 | void set(int bitIndex, boolean value) 这种方法指定索引到指定设置该位的值。 |
| 22 | void set(int fromIndex, int toIndex) 此方法设置的位将指定的fromIndex（包括）到指定的toIndex（不包括）为true。 |
| 23 | void set(int fromIndex, int toIndex, boolean value) 此方法从指定的fromIndex（包括）设置位到指定的toIndex（不包括）到指定的值。 |
| 24 | int size() 此方法返回实际使用此BitSet表示位值的空间的比特数。 |
| 25 | String toString() 此方法返回此位set的字符串表示形式。 |
| 26 | void xor(BitSet set) 此方法执行此位的逻辑异或设置该位设置参数。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# Java.util.Calendar类 - Java.util包

**java.util.calendar** 类是一个抽象类，它提供了与在某一特定时刻和一组日历字段如YEAR，MONTH，DAY_OF_MONTH，HOUR等，并为操作日历字段，如获取的日期转换方法下周。以下是关于日历的要点：

- 这个类还提供了额外的字段和实施具体的日历系统外面包的方法。
- 日历定义了某些日历字段返回值的范围。

## 类的声明

以下是java.util.Calendar类的声明：

```
public abstract class Calendar
   extends Object
      implements Serializable, Cloneable, Comparable<Calendar>
```

## 字段

以下是java.util.Calendar类中的字段：

- static int ALL_STYLES -- 这个风格说明符getDisplayNames指示所有样式的名称，如"January" 和"Jan"。
- static int AM -- 这是AM_PM字段指示当天的期限从半夜到中午前的值。
- static int AM_PM -- 这个字段数get和set的指示HOUR是中午之前或之后。
- static int APRIL -- 这个MONTH字段表示第四个月当年在格里高利历和罗马儒略历中的值。
- protected boolean areFieldsSet -- 为true，如果fields[] 与当前设置的时间同步。
- static int AUGUST -- 这是一个MONTH字段指示第八个年度在格里高利历和罗马儒略历中的值。
- static int DATE -- 这是字段数为get 和 set，表示该月的日期。
- static int DAY_OF_MONTH -- 这是字段数为get 和 set，表示该月的日期。
- static int DAY_OF_WEEK -- 这是字段数get和set表示星期几。
- static int DAY_OF_WEEK_IN_MONTH -- 这是字段数为get和set，指示当前月中的序数星期。

- static int DAY_OF_YEAR -- 这是字段数为get和set，表示本年度内天数。
- static int DECEMBER -- 这是一个MONTH字段指示腊月当年在格里高利历和罗马儒略历中的值。
- static int DST_OFFSET -- 这是字段数get和set的指示夏令时以毫秒为单位的偏移。
- static int ERA -- 这是字段数为get和set显示的时代，例如，在儒略历中的AD或BC。
- static int FEBRUARY -- 这是一个MONTH字段指示第二个年度在格里高利历和罗马儒略历中的值。
- static int FIELD_COUNT -- 这是通过获取确认和设置不同的字段的数量。
- protected int[] fields -- 这是当前设置的时间为这个日历的日历字段值。
- static int FRIDAY -- 这是DAY_OF_WEEK字段的说明周五的值。
- static int HOUR -- 这是字段数为get和set，表示上午或下午的时间。
- static int HOUR_OF_DAY -- 这是字段数get和set表示一天中的小时。
- protected boolean[] isSet -- 这是判断该日历某一指定日历字段设置该标志。
- protected boolean isTimeSet -- 这是true如果当时的时间值是有效的。
- static int JANUARY -- 这是一个MONTH字段表示第一个月，一年的公历和罗马儒略历中的值。
- static int JULY -- 这是一个MONTH字段表示第七个月当年在格里高利历和罗马儒略历中的值。
- static int JUNE -- 这是一个MONTH字段表明了第六个月，一年的公历和罗马儒略历中的值。
- static int LONG -- 这是风格说明符getDisplayName和getDisplayNames表示长的名字，如“January”。
- static int MARCH -- 这是一个MONTH字段指示第三个年度在格里高利历和罗马儒略历中的值。
- static int MAY -- 这是一个MONTH字段表示第五个月当年在格里高利历和罗马儒略历中的值。
- static int MILLISECOND -- 这是字段数 get 和 set 指示第二内毫秒。
- static int MINUTE -- 这是字段数 get 和 set 指示一小时中的分钟。
- static int MONDAY -- 这是DAY_OF_WEEK字段的说明周一的值。
- static int MONTH -- 这是字段数为get 和 set，指示一个月。

- static int NOVEMBER -- 这是一个MONTH字段指示第十一个月当年在格里高利历和罗马儒略历中的值。
- static int OCTOBER -- 这是一个MONTH字段指示第十一个月，一年中的格里高利历和罗马儒略历中的值。
- static int PM -- 这是AM_PM字段中的指示当天的期限从中午到午夜前的值。
- static int SATURDAY -- 这是DAY_OF_WEEK字段的指示周六的值。
- static int SECOND -- 这是字段数get和set指示一分钟中的秒。
- static int SEPTEMBER -- 这是一个MONTH字段表示第九个月当年在格里高利历和罗马儒略历中的值。
- static int SHORT -- 这是风格说明符getDisplayName和getDisplayNames显示一个简短的名称，如"Jan"。
- static int SUNDAY -- 这是DAY_OF_WEEK字段的说明周日的值。
- static int THURSDAY -- 这是DAY_OF_WEEK字段的说明周四的值。
- protected long time -- 这是当前设置的时间，这个日历，以毫秒表示1970年1月1日，0:00:00 GMT之后。
- static int TUESDAY -- 这是DAY_OF_WEEK字段的说明星期二的值。
- static int UNDECIMBER -- 这是一个MONTH字段指示一年第十三个月的值。
- static int WEDNESDAY -- 这是DAY_OF_WEEK字段说明周三的值。
- static int WEEK_OF_MONTH -- 这是字段数为GET和SET，指示当前月中的星期数。
- static int WEEK_OF_YEAR -- 这是字段数为GET和SET，表示在本年度内的周数。 。
- static int YEAR -- 这是字段数get和set表示年份。
- static int ZONE_OFFSET -- 这是字段数为GET和SET指示原与GMT以毫秒为单位的偏移。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected Calendar()** 这个构造函数构造一个日历使用默认时区和语言环境。 |
| 2 | **protected Calendar(TimeZone zone, Locale aLocale)** 这个构造函数构造一个日历指定的时区和语言环境。 |

# 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | abstract void add(int field, int amount) 此方法添加或减去指定的时间量，以给定日历字段，基于日历的规则。 |
| 2 | boolean after(Object when) 这个方法返回当前日历是否代表在指定Object表示的时间之后的时间。 |
| 3 | boolean before(Object when) 这个方法返回当前日历是否代表在指定Object表示的时间之前的时间。 |
| 4 | void clear() 此方法设置此Calendar的所有日历字段值和时间值（毫秒从历元至偏移量）未定义。 |
| 5 | void clear(int field) 此方法设置给定日历字段值和本日历不确定的时间值（毫秒从历元至偏移量）。 |
| 6 | Object clone() 此方法创建并返回此对象的一个副本。 |
| 7 | int compareTo(Calendar anotherCalendar) 这个方法比较两个Calendar对象表示的时间值（从历元至毫秒偏移量）。 |
| 8 | protected void complete() 此方法填充在日历字段中所有未设置的字段。 |
| 9 | protected abstract void computeFields() 这种方法的当前毫秒时间值时间日历fields[]字段值转换。 |
| 10 | protected abstract void computeTime() 这种方法在fields[]到毫秒的时间值时将转换当前日历字段值。 |
| 11 | boolean equals(Object obj) 这个方法这个日历比较指定的对象。 |
| 12 | int get(int field) 此方法返回给定日历字段的值。 |
| 13 | int getActualMaximum(int field) 此方法返回指定日历字段可能拥有的最大值，鉴于此Calendar时间值。 |
| 14 | int getActualMinimum(int field) 此方法返回指定日历字段可能拥有的最小值，鉴于此Calendar时间值。 |
| 15 | static Locale[] getAvailableLocales() 此方法返回所有语言环境，它由此类的getInstance方法可为之返回本地化实例的数组。 |
| 16 | String getDisplayName(int field, int style, Locale locale) 此方法返回的日历字段值在给定的风格和语言环境的字符串表示形式。 |
| 17 | Map<String,Integer> getDisplayNames(int field, int style, Locale locale) 这个方法返回一个Map包含日历字段在给定的风格和语言环境和相应的字段值的所有名称。 |
| 18 | int getFirstDayOfWeek() 这种方法得到一周的第一天是什么;例如，在美 |

| | |
|---|---|
| 18 | 国为SUNDAY，在法国为MONDAY。 |
| 19 | abstract int getGreatestMinimum(int field) 此方法返回此Calendar实例给定日历字段的最高的最小值。 |
| 20 | static Calendar getInstance() 使用默认时区和语言环境这种方法获得一个日历。 |
| 21 | static Calendar getInstance(Locale aLocale) 使用默认时区和指定的区域设置此方法获取一个日历。 |
| 22 | static Calendar getInstance(TimeZone zone) 使用指定的时区和默认语言环境这种方法得到一个日历。 |
| 23 | static Calendar getInstance(TimeZone zone, Locale aLocale) 这种方法得到一个日历指定的时区和语言环境。 |
| 24 | abstract int getLeastMaximum(int field) 此方法返回此Calendar实例给定日历字段的最低的最大值。 |
| 25 | abstract int getMaximum(int field) 此方法返回此Calendar实例给定日历字段的最大值。 |
| 26 | int getMinimalDaysInFirstWeek() 这种方法得到什么在今年的第一个星期所需的最少天数;例如，如果第一周被定义为一个包含在第一个月的一年的第一天，此方法返回1。 |
| 27 | abstract int getMinimum(int field) 此方法返回此Calendar实例给定日历字段的最小值。 |
| 28 | Date getTime() 此方法返回表示此Calendar的时间值（从历元至"毫秒偏移量）的Date对象。 |
| 29 | long getTimeInMillis() 该方法以毫秒为单位返回此Calendar的时间值。 |
| 30 | TimeZone getTimeZone() 这种方法获取的时区。 |
| 31 | int hashCode() 此方法返回此日历的哈希码。 |
| 32 | protected int internalGet(int field) 此方法返回给定日历字段的值。 |
| 33 | boolean isLenient() 这个方法告诉日期/时间的解释是否是宽松的。 |
| 34 | boolean isSet(int field) 此方法可确定给定日历字段的值集，包括案件的值被设置由一个get方法调用触发内部字段计算。 |
| 35 | abstract void roll(int field, boolean up) 此方法添加或减去（上/下）的时候一个单元在给定的时间字段不更改更大的字段。 |
| 36 | void roll(int field, int amount) 此方法将指定的（签署）金额至指定日历字段不更改更大的字段。 |
| 37 | void set(int field, int value) 此方法设置给定日历字段为给定值。 |
| | void set(int year, int month, int date) 此方法设置为日历字段的值 YEAR, |

| | |
|---|---|
| | MONTH, and DAY_OF_MONTH.. |
| 39 | void set(int year, int month, int date, int hourOfDay, int minute) 此方法设置为日历字段的值YEAR, MONTH, DAY_OF_MONTH, HOUR_OF_DAY, 和 MINUTE. |
| 40 | void set(int year, int month, int date, int hourOfDay, int minute, int second) 此方法设置的字段的值YEAR, MONTH, DAY_OF_MONTH, HOUR, MINUTE, 和SECOND. |
| 41 | void setFirstDayOfWeek(int value) 此方法设置一周的第一天是什么;例如，在美国为SUNDAY，在法国为MONDAY。 |
| 42 | void setLenient(boolean lenient) 此方法规定日期/时间的解释是否是宽松的。 |
| 43 | void setMinimalDaysInFirstWeek(int value) 此方法设置在哪一年中第一个星期所需的最少天数;例如，如果在第一周被定义为一个包含在第一个月的一年的第一天，传值调用这个方法。 |
| 44 | void setTime(Date date) 此方法设置此Calendar时间与给定的日期。 |
| 45 | void setTimeInMillis(long millis) 这个方法从给定的long值设置此Calendar的当前时间。 |
| 46 | void setTimeZone(TimeZone value) 此方法使用给定的时区值设置时区。 |
| 47 | String toString() 此方法返回此日历的字符串表示形式。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# Java.util.Collections类 - Java.util包

**java.util.Collections**类专门由操作上或返回集合的静态方法。以下是关于集合的要点：

- 它包含在collection上，“包装”，它会返回一个新的集合由一个指定的集合操作支持多态算法。
- 这个类的方法都抛出NullPointerException如果提供给他们的集合或类对象为null。

## 类的声明

以下是java.util.Collections类的声明：

```
public class Collections
   extends Object
```

## 字段域

以下是java.util.Collections类中的字段：

- static List EMPTY_LIST -- 这是一个空的列表(不可变)。
- static Map EMPTY_MAP -- 这是空映射(不可变)。
- static Set EMPTY_SET -- 这是空集(不可变)。

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static <T> boolean addAll(Collection<? super T> c, T... elements) 这个方法会将所有指定的元素到指定的集合。 |
| 2 | static <T> Queue<T> asLifoQueue(Deque<T> deque) 此方法返回一个deque的视图，作为一个后进先出(LIFO)队列。 |
| 3 | static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key) 此方法搜索指定列表，使用二进制搜索算法来指定对象。 |
| 4 | static <T> int binarySearch(List<? extends T> list, T key, Comparator<? super T< c) 此方法搜索指定列表，使用二进制搜索算法来指定对象。 |

| | |
|---|---|
| 5 | static <E> Collection<E> checkedCollection(Collection<E> c, Class<E> type) 此方法返回指定集合的动态类型安全视图。 |
| 6 | static <E> List<E> checkedList(List<E> list, Class<E> type) 此方法返回指定列表的一个动态类型安全视图。 |
| 7 | static <K,V> Map<K,V> checkedMap(Map<K,V> m, Class<K> keyType, Class<V> valueType) 此方法返回指定映射的一个动态类型安全视图。 |
| 8 | static <E> Set<E> checkedSet(Set<E> s, Class<E> type) 此方法返回指定set的一个动态类型安全视图。 |
| 9 | static <K,V> SortedMap<K,V> checkedSortedMap(SortedMap<K,V> m, Class<K> keyType, Class<V> valueType) 此方法返回指定有序映射的一个动态类型安全视图。 |
| 10 | static <E> SortedSet<E> checkedSortedSet(SortedSet<E> s, Class<E> type) 此方法返回指定有序set的一个动态类型安全视图。 |
| 11 | static <T> void copy(List<? super T> dest, List<? extends T> src) 这个方法会将所有从一个列表中的元素到另一个。 |
| 12 | static boolean disjoint(Collection<?> c1, Collection<?> c2) 如果两个指定collection中没有相同的元素此方法返回true。 |
| 13 | static <T> List<T> emptyList() 此方法返回空列表（不可变）。 |
| 14 | static <K,V> Map<K,V> emptyMap() 此方法返回空映射（不可变）。 |
| 15 | static <T> Set<T> emptySet() 此方法返回空集（不可变）。 |
| 16 | static <T> Enumeration<T> enumeration(Collection<T> c) 此方法返回一个枚举在指定的collection。 |
| 17 | static <T> void fill(List<? super T> list, T obj) 此方法替换所有指定的列表中具有指定元素的元素。 |
| 18 | static int frequency(Collection<?> c, Object o) 此方法返回指定元素集合等于指定对象的数量。 |
| 19 | static int indexOfSubList(List<?> source, List<?> target) 此方法返回指定目标列表中第一次出现的起始位置的指定源列表中，或者-1，如果没有发生。 |
| 20 | static int lastIndexOfSubList(List<?> source, List<?> target) 此方法返回指定目标列表的最后出现的起始位置指定源列表中，或者-1，如果没有发生。 |
| 21 | static <T> ArrayList<T> list(Enumeration<T> e) 此方法返回包含由指定枚举它们是由枚举返回的顺序返回元素的数组列表。 |
| 22 | static <T extends Object & Comparable<? super T> >T max(Collection<? extends T> coll) 此方法返回给定collection的最大元 |

| | |
|---|---|
| | 素，根据其元素的自然顺序。 |
| 23 | static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp) 此方法返回给定collection的最大元素，根据诱发指定的比较器的顺序。 |
| 24 | static <T extends Object & Comparable<? super T>>T min(Collection<? extends T> coll) 这个方法返回给定collection的最小元素，根据其元素的自然顺序。 |
| 25 | static <T> T min(Collection<? extends T> coll, Comparator<? super T> comp) 此方法返回给定collection的最小元素，根据诱发指定的比较器的顺序。 |
| 26 | static <T> List<T> nCopies(int n, T o) 此方法返回一个不可变列表组成的n个拷贝的指定对象。 |
| 27 | static <E> Set<E> newSetFromMap(Map<E,Boolean> map) 此方法返回一组由指定映射支持。 |
| 28 | static <T> boolean replaceAll(List<T> list, T oldVal, T newVal) 这种方法取代了与另一个列表中的一个指定值的所有匹配。 |
| 29 | static void reverse(List<?> list) 这种方法将反转元素的顺序指定列表 |
| 30 | static <T> Comparator<T> reverseOrder() 此方法返回一个比较器，它强行上实现Comparable接口的对象的集合的自然顺序相反。 |
| 31 | static <T> Comparator<T> reverseOrder(Comparator<T> cmp) 此方法返回一个比较器，它强行指定比较器的反向排序。 |
| 32 | static void rotate(List<?> list, int distance) 此方法通过指定的距离旋转指定列表中的元素。 |
| 33 | static void shuffle(List<?> list) 此方法随机进行置换使用随机性的默认源中指定的列表。 |
| 34 | static void shuffle(List<?> list, Random rnd) 此方法随机排列使用随机指定源指定的列表。 |
| 35 | static <T> Set<T> singleton(T o) 此方法返回一个不可变的集只包含指定对象。 |
| 36 | static <T> List<T> singletonList(T o) 此方法返回一个只包含指定对象的不可变列表。 |
| 37 | static <K,V> Map<K,V> singletonMap(K key, V value) 此方法返回一个不可变的映射，映射只有指定的键为指定的值。 |
| 38 | static <T extends Comparable<? super T>> void sort(List<T> list) 这种方法对指定列表按升序顺序，根据其元素的自然顺序。 |
| 39 | static <T> void sort(List<T> list, Comparator<? super T> c) 根据诱导由指定比较器的顺序这种方法排序指定列表。 |

| | |
|---|---|
| 40 | static void swap(List<?> list, int i, int j) 这个方法在交换指定列表中指定位置的元素。 |
| 41 | static <T> Collection<T> synchronizedCollection(Collection<T> c) 这个方法返回一个同步的（线程安全的）集合的指定集合的支持。 |
| 42 | static <T> List<T> synchronizedList(List<T> list) 此方法返回由指定列表支持的同步（线程安全的）列表。 |
| 43 | static <K,V> Map<K,V> synchronizedMap(Map<K,V> m) 这个方法返回一个同步的（线程安全）地图由指定映射支持。 |
| 44 | static <T> Set<T> synchronizedSet(Set<T> s) 这个方法返回一个同步的（线程安全的）集由指定set支持。 |
| 45 | static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m) 这个方法返回一个同步的（线程安全的）有序映射所指定的有序映射支持。 |
| 46 | static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s) 这个方法返回一个同步的（线程安全的）有序set由指定的有序set支持。 |
| 47 | static <T> Collection<T> unmodifiableCollection(Collection<? extends T> c) 此方法返回指定collection的不可修改视图。 |
| 48 | static <T> List<T> unmodifiableList(List<? extends T> list) 此方法返回指定列表的不可修改视图。 |
| 49 | static <K,V> Map<K,V> unmodifiableMap(Map<? extends K,? extends V> m) 此方法返回指定映射的不可修改视图。 |
| 50 | static <T> Set<T> unmodifiableSet(Set<? extends T> s) 此方法返回指定集合的不可修改视图。 |
| 51 | static <K,V> SortedMap<K,V> unmodifiableSortedMap(SortedMap<K,? extends V> m) 此方法返回指定有序映射的不可修改视图 |
| 52 | static <T> SortedSet<T> unmodifiableSortedSet(SortedSet<T> s) 此方法返回指定有序集合不可修改视图。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# Java.util.Currency类 - Java.util包

**java.util.Currency** 类代表货币。下面是有关货币的要点：

- 货币是通过它们的ISO4217货币代码。
- 类的设计，使从未有任何给定的货币多于一个Currency实例，这是背后没有公共的构造函数的原因。

## 类的声明

以下是声明了java.util.Currency类：

```
public final class Currency
   extends Object
      implements Serializable
```

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | String getCurrencyCode() 此方法获取此货币的ISO4217货币代码。 |
| 2 | int getDefaultFractionDigits() 此方法得到的分数与此货币使用的缺省小数位数。 |
| 3 | static Currency getInstance(Locale locale) 此方法返回给定语言环境的国家的货币实例。 |
| 4 | static Currency getInstance(String currencyCode) 此方法返回给定货币代码的Currency实例。 |
| 5 | String getSymbol() 此方法获取此货币的符号为默认语言环境。 |
| 6 | String getSymbol(Locale locale) 此方法获取此货币的符号为指定的语言环境。 |
| 7 | String toString() 此方法返回此货币的ISO4217货币代码。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.Date类 - Java.util包

**java.util.Date**类表示某一特定时刻，精确到毫秒。

## 类的声明

以下是java.util.Date类的声明：

```
public class Date
   extends Object
      implements Serializable, Cloneable, Comparable<Date>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Date()** 这个构造函数分配一个Date对象并将它初始化，使它表示其被分配的时间，精确到毫秒。 |
| 2 | **Date(long date)** 这个构造函数分配一个Date对象并初始化它代表指定的毫秒数，因为被称为“纪元”，即1970年1月1日00:00:00 GMT标准基准时间。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean after(Date when) 此方法测试，此日期是否在指定日期之后。 |
| 2 | boolean before(Date when) 此方法测试，此日期是否在指定日期之前。 |
| 3 | Object clone() 此方法返回此对象的一个副本。 |
| 4 | int compareTo(Date anotherDate) 此方法比较两个日期的顺序。 |
| 5 | boolean equals(Object obj) 此方法比较两个日期是否相等。 |
| 6 | long getTime() 此方法返回自1970年1月1日00:00:00 GMT此Date对象表示的毫秒数。 |
| 7 | int hashCode() 此方法返回此对象的哈希码值。 |
| 8 | void setTime(long time) 此方法设置此Date对象1970年1月1日00:00:00 GMT以后，代表一个时间点time毫秒。 |
| 9 | String toString() 此方法此Date对象转换为形式的字符串。 |

## 方法继承

这个类继承自以下类方法：

- java.util.Object

# java.util.Dictionary类 - Java.util包

**java.util.Dictionary** 类是任何类的抽象父类，如哈希表，其中键映射到相应的值。以下是关于字典的要点：

- 在这个类中的每个键，每个值都是一个对象。
- 在这个类的对象的每个键与最多只有一个值相关联。

## 类声明

以下是java.util.Dictionary类的声明：

```
public abstract class Dictionary<K,V>
   extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Dictionary()** 这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | abstract Enumeration<V> elements() 这种方法在这个字典中返回枚举值。 |
| 2 | abstract V get(Object key) 此方法返回到该键所映射在这个字典中的值。 |
| 3 | abstract boolean isEmpty() 如果这此字典是否有键映射到的值的方法测试。 |
| 4 | abstract Enumeration<K> keys() 此方法返回这个字典中的键的枚举。 |
| 5 | abstract V put(K key, V value) 这种方法映射到指定键在此字典中指定的值。 |
| 6 | abstract V remove(Object key) 这个方法从这个字典中删除键(及其相应的值)。 |
| 7 | abstract int size() 此方法在此字典返回条目的数量(不同的键)。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.EnumMap类 - Java.util包

**java.util.EnumMap** 类是一种专门Map实现与枚举键的使用。以下是有关EnumMap要点：

- 所有在枚举映射的键都必须来自所指定，或明或暗地，创建映射的时候一个枚举类型。
- 枚举映射保持在其键的自然顺序。
- EnumMap是不同步的。如果多个线程同时访问一个枚举映射并发和线程中的至少一个修改的映射，它应该保持外部同步。

## 类声明

以下是java.util.EnumMap类的声明：

```
public class EnumMap<K extends Enum<K>,V>
   extends AbstractMap<K,V>
      implements Serializable, Cloneable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **EnumMap(Class<K> keyType)** 此构造函数创建具有指定键类型的空枚举映射。 |
| 2 | **EnumMap(EnumMap<K,? extends V> m)** 此构造函数创建具有相同的键类型与指定枚举映射的枚举映射，最初包含相同的映射关系(如果有的话)。 |
| 3 | **EnumMap(Map<K,? extends V> m)** 此构造函数从指定的映射表进行初始化创建一个枚举映射。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void clear() 此方法从此映射删除所有映射。 |
| 2 | EnumMap<K,V> clone() 此方法返回此枚举映射的浅表副本。 |
| 3 | boolean containsKey(Object key) 如果此映射包含指定键的映射此方法返回true。 |
| 4 | boolean containsValue(Object value) 如果此映射一个或多个键映射到指定值，该方法返回true。 |
| 5 | Set<Map.Entry<K,V>> entrySet() 此方法返回此映射中包含的映射关系的Set视图。 |
| 6 | boolean equals(Object o) 此方法将指定对象与此映射比较是否相等。 |
| 7 | V get(Object key) 这个方法返回指定键所映射的值，或者null，如果此映射不包含该键。 |
| 8 | Set<K> keySet() 此方法返回此映射中包含的键的Set视图。 |
| 9 | V put(K key, V value) 这种方法与关联此映射中的指定键指定的值。 |
| 10 | void putAll(Map<? extends K,? extends V> m) 这种方法将所有从指定映射此映射中的映射。 |
| 11 | V remove(Object key) 此方法从该映射删除映射为这个键，如果存在的话。 |
| 12 | int size() 此方法返回键- 值映射关系在映射的大小。 |
| 13 | Collection<V> values() 此方法返回此映射中包含的值的Collection视图。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractMap
- java.util.Object

# java.util.EnumSet类 - Java.util包

**java.util.EnumSet** 类是一种专门设置实现与枚举类型的使用。以下是关于EnumSet这重要的几点：

- 所有的枚举set的元素必须来自指定的，或明或暗地，在创建时设置一个枚举类型。
- 枚举set在内部表示为位向量。
- EnumSet是不同步的。如果多个线程同时访问一个枚举同时设置，并且至少有一个线程修改的设置，它应该保持外部同步。

## 类声明

以下是java.util.EnumSet类的声明：

```
public abstract class EnumSet<E extends Enum<E>>
  extends AbstractSet<E>
     implements Cloneable, Serializable
```

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static <E extends Enum<E>> EnumSet<E> allOf(Class<E> elementType) 此方法创建一个包含所有在指定元素类型的元素的枚举set。 |
| 2 | EnumSet<E> clone() 此方法返回这个集合的一个副本。 |
| 3 | static <E extends Enum<E>> EnumSet<E> complementOf(EnumSet<E> s) 此方法创建一个枚举设置相同的元素类型与指定枚举set，最初包含此类型的所有未包含指定集合中的元素。 |
| 4 | static <E extends Enum<E>> EnumSet<E> copyOf(Collection<E> c) 此方法创建一个枚举集从指定集合初始化。 |
| 5 | static <E extends Enum<E>> EnumSet<E> copyOf(EnumSet<E> s) 此方法创建一个枚举设置相同的元素类型与指定枚举set，最初包含相同的元素（如果有的话）。 |
| 6 | static <E extends Enum<E>> EnumSet<E> noneOf(Class<E> elementType) 此方法创建一个空的枚举set具有指定元素类型。 |
| 7 | static <E extends Enum<E>> EnumSet<E> of(E e) 此方法创建一个最初包含指定元素的枚举set。 |
| 8 | static <E extends Enum<E>> EnumSet<E> of(E first, E... rest) 此方法创建一个最初包含指定元素的枚举set。 |
| 9 | static <E extends Enum<E>> EnumSet<E> of(E e1, E e2) 此方法创建一个最初包含指定元素的枚举set。 |
| 10 | static <E extends Enum<E>> EnumSet<E> of(E e1, E e2, E e3) 此方法创建一个最初包含指定元素的枚举set。 |
| 11 | static <E extends Enum<E>> EnumSet<E> of(E e1, E e2, E e3, E e4) 此方法创建一个最初包含指定元素的枚举set。 |
| 12 | static <E extends Enum<E>> EnumSet<E> of(E e1, E e2, E e3, E e4, E e5) 此方法创建一个最初包含指定元素的枚举set。 |
| 13 | static <E extends Enum<E>> EnumSet<E> range(E from, E to) 此方法创建最初包含由两个指定端点所定义范围的元素的枚举set。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractSet
- java.util.AbstractCollection
- java.util.Object

- java.util.Set

# java.util.Formatter类 - Java.util包

**java.util.Formatter** 类提供了布局合理性和对齐方式，常见格式为数字，字符串和日期/时间数据，以及语言环境的输出的支持。以下是关于格式化要点：

- 格式化并不一定是安全的多线程访问。线程安全是可选的，在这个类方法由用户自已定义。

## 类声明

以下是java.util.Formatter类的声明：

```
public final class Formatter
   extends Object
      implements Closeable, Flushable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Formatter()** 这个构造函数构造一个新的格式化。 |
| 2 | **Formatter(Appendable a)** 这个构造函数构造一个新的格式化与指定的目标。 |
| 3 | **Formatter(Appendable a, Locale l)** 这个构造函数构造一个新的格式化与指定的目标和语言环境。 |
| 4 | **Formatter(File file)** 这个构造函数构造一个新的格式化与指定的文件。 |
| 5 | **Formatter(File file, String csn)** 这个构造函数构造一个新的格式化与指定文件和字符集。 |
| 6 | **Formatter(File file, String csn, Locale l)** 这个构造函数构造一个新的格式化与指定的文件，字符集和语言环境。 |
| 7 | **Formatter(Locale l)** 这个构造函数构造一个新的格式化与指定的语言环境。 |
| 8 | **Formatter(OutputStream os)** 这个构造函数构造一个新的格式化与指定的输出流。 |
| 9 | **Formatter(OutputStream os, String csn)** 这个构造函数构造一个新的格式化与指定的输出流和字符集。 |
| 10 | **Formatter(OutputStream os, String csn, Locale l)** 这个构造函数构造一个新的格式化与指定的输出流，字符集和语言环境。 |
| 11 | **Formatter(PrintStream ps)** 这个构造函数构造一个新的格式化与指定的打印流。 |
| 12 | **Formatter(String fileName)** 这个构造函数构造一个新的格式化与指定的文件名。 |
| 13 | **Formatter(String fileName, String csn)** 这个构造函数构造一个新的格式化与指定的文件名和字符集。 |
| 14 | **Formatter(String fileName, String csn, Locale l)** 这个构造函数构造一个新的格式化与指定的文件名，字符集和语言环境。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭此格式化程序。 |
| 2 | void flush() 这个方法刷新此格式化程序。 |
| 3 | Formatter format(Locale l, String format, Object... args) 此方法写入一个格式化字符串使用指定的语言环境，格式字符串和参数，此对象的目标。 |
| 4 | Formatter format(String format, Object... args) 此方法写入一个格式化字符串使用指定格式字符串和参数此对象的目标。 |
| 5 | IOException ioException() 此方法返回的最后一个IOException异常被抛出此格式化程序的附加。 |
| 6 | Locale locale() 这个方法返回locale这个格式化的结构设置。 |
| 7 | Appendable out() 此方法返回的目的地的输出。 |
| 8 | String toString() 此方法返回调用toString()方法在目的地的输出结果。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.GregorianCalendar类 - Java.util包

java.util.GregorianCalendar 类是Calendar的一个具体子类，提供用于世界上大多数国家的标准日历系统。以下是关于GregorianCalendar的要点：

- 它是同时支持朱利安和公历系统的一个不连续，这相当于在默认情况下，当公历被提起的公历日期的支持混合日历。
- 儒略历指定闰年每四年，而公历省略世纪十年哪些不是被400整除。

## 类声明

以下是java.util.GregorianCalendar类的声明：

```
public class GregorianCalendar
   extends Calendar
```

## 字段域

以下是java.util.GregorianCalendar类中的字段：

- static int AD -- 这是ERA字段的指示共同的时代(公元)，也被称为CE的值。
- static int BC -- 这是时代字段的值，表示共同的时代之前的期间(公元前)，也被称为BCE。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **GregorianCalendar()** 这个构造使用当前时间的默认时区与默认语言环境的默认的GregorianCalendar。 |
| 2 | **GregorianCalendar(int year, int month, int dayOfMonth)** 这个构造一个GregorianCalendar与给定日期的默认时区设置默认语言环境。 |
| 3 | **GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute)** 这个构造一个GregorianCalendar用给定的日期和时间设置为与默认语言环境的默认时区。 |
| 4 | **GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second)** 这个构造一个GregorianCalendar用给定的日期和时间设置为与默认语言环境的默认时区。 |
| 5 | **GregorianCalendar(Locale aLocale)** 这构建了基于当前时间与给定语言环境的默认时区一个GregorianCalendar。 |
| 6 | **GregorianCalendar(TimeZone zone)** 这构建了基于当前时间与默认语言环境的给定时区一个GregorianCalendar。 |
| 7 | **GregorianCalendar(TimeZone zone, Locale aLocale)** 这构建了基于当前时间与给定语言环境的给定时区一个GregorianCalendar。 |

# 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void add(int field, int amount) 此方法将指定(有符号的)时间量，以给定日历字段，根据日历的规则。 |
| 2 | Object clone() 此方法创建并返回此对象的一个副本。 |
| 3 | protected void computeFields() 这种方法的时间值(毫秒从历元至偏移量)，以日历字段值转换。 |
| 4 | protected void computeTime() 这种方法日历字段值转换为时间值(从历元至毫秒偏移量)。 |
| 5 | boolean equals(Object obj) This method compares this GregorianCalendar to the specified Object. |
| 6 | int getActualMaximum(int field) This method returns the maximum value that this calendar field could have, taking into consideration the given time value and the current values of the getFirstDayOfWeek, getMinimalDaysInFirstWeek, getGregorianChange and getTimeZone methods. |
|  | int getActualMinimum(int field) This method returns the minimum value that this calendar field could have, taking into consideration the given |

| | |
|---|---|
| 7 | time value and the current values of the getFirstDayOfWeek, getMinimalDaysInFirstWeek, getGregorianChange and getTimeZone methods. |
| 8 | int getGreatestMinimum(int field) This method returns the highest minimum value for the given calendar field of this GregorianCalendar instance. |
| 9 | Date getGregorianChange() This method gets the Gregorian Calendar change date. |
| 10 | int getLeastMaximum(int field) This method returns the lowest maximum value for the given calendar field of this GregorianCalendar instance. |
| 11 | int getMaximum(int field) This method returns the maximum value for the given calendar field of this GregorianCalendar instance. |
| 12 | int getMinimum(int field) This method returns the minimum value for the given calendar field of this GregorianCalendar instance. |
| 13 | TimeZone getTimeZone() This method gets the time zone. |
| 14 | int hashCode() This method generates the hash code for this GregorianCalendar object. |
| 15 | boolean isLeapYear(int year) This method determines if the given year is a leap year. |
| 16 | void roll(int field, boolean up) This method adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields. |
| 17 | void roll(int field, int amount) This method adds a signed amount to the specified calendar field without changing larger fields. |
| 18 | void setGregorianChange(Date date) This method sets the GregorianCalendar change date. |
| 19 | setTimeZone(TimeZone zone) This method sets the time zone with the given time zone value. |

# 方法继承

这个类从以下类继承的方法：

- java.util.Calendar
- java.util.Object

# java.util.HashMap类 - Java.util包

**java.util.HashMap** 类是基于哈希表的Map接口的实现。以下是关于HashMap的要点：

- 这个类不保证为向映射的迭代顺序;特别是，它不保证该顺序将继续随时间恒定。
- 这个类允许null值和null键。

## 类声明

以下是java.util.HashMap类的声明：

```
public class HashMap<K,V>
   extends AbstractMap<K,V>
       implements Map<K,V>, Cloneable, Serializable
```

## 参数

以下是java.util.HashMap类中的参数：

- K -- 这是映射保持的键的类型。
- V -- 这是映射值的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **HashMap()** 此构造一个空的HashMap具有默认初始容量（16）和默认加载因子（0.75）。 |
| 2 | **HashMap(Collection<? extends E> c)** 此构造一个空的HashMap具有指定的初始容量和默认加载因子（0.75）。 |
| 3 | **HashMap(int initialCapacity, float loadFactor)** 此构造一个空的HashMap具有指定的初始容量和加载因子。 |
| 4 | **HashMap(Map<? extends K,? extends V> m)** 这种构造一个新的HashMap中使用相同的映射关系与指定映射。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | void clear() 此方法删除所有来自此映射中的映射。 |
| 2 | Object clone() 此方法返回此HashMap实例的浅表副本，键和值本身不被复制。 |
| 3 | boolean containsKey(Object key) 如果此映射包含指定键的映射此方法返回true。 |
| 4 | boolean containsValue(Object value) 如果此映射一个或多个键映射到指定值，该方法返回true。 |
| 5 | Set<Map.Entry<K,V>> entrySet() 此方法返回此映射中包含的映射关系的Set视图。 |
| 6 | V get(Object key) 这个方法返回指定键所映射的值，或者null，如果此映射不包含该键。 |
| 7 | boolean isEmpty() 如果此映射不包含键 - 值映射，此方法返回true。 |
| 8 | Set<K> keySet() 此方法返回此映射中包含的键的Set视图。 |
| 9 | V put(K key, V value) 这种方法与关联此映射中的指定键指定的值。 |
| 10 | void putAll(Map<? extends K,? extends V> m) 这个方法会将所有从指定映射此映射中的映射。 |
| 11 | V remove(Object key) 此方法删除映射对于指定的键从该映射，如果存在的话。 |
| 12 | int size() 此方法返回键 - 值映射关系在这个映射的数量。 |
| 13 | Collection<V> values() 此方法返回此映射中包含的值的Collection视图。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractMap
- java.util.Object
- java.util.Map

# java.util.HashSet类 - Java.util包

java.util.HashSet 类实现Set接口，由哈希表支持。以下是关于HashSet的要点：

- 这个类不做任何担保，以集合的迭代顺序;特别是，它不保证该顺序将继续随时间恒定。
- 此类允许null元素。

## 类声明

以下是java.util.HashSet类的声明：

```
public class HashSet<E>
   extends AbstractSet<E>
      implements Set<E>, Cloneable, Serializable
```

## 参数

以下是java.util.HashSet类的参数：

- E -- 这是此set保留元素的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **HashSet()** 这种构造一个新的空集;其底层HashMap实例具有默认初始容量(16)和加载因子(0.75)。 |
| 2 | **HashSet(Collection<? extends E> c)** 这个构造一个包含指定集合中的元素的新集。 |
| 3 | **HashSet(int initialCapacity)** 这种构造一个新的空集;其底层HashMap实例具有指定的初始容量和默认加载因子(0.75)。 |
| 4 | **HashSet(int initialCapacity, float loadFactor)** 这种构造一个新的空集;其底层HashMap实例具有指定的初始容量和指定加载因子。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean add(E e) 此方法将指定的元素添加到此集合，如果它是不存在的。 |
| 2 | void clear() 此方法删除这个集合中的所有元素。 |
| 3 | Object clone() 此方法返回此HashSet实例的浅表副本，元素本身没有复制。 |
| 4 | boolean contains(Object o) 如果此set包含指定的元素，此方法返回true。 |
| 5 | boolean isEmpty() 如果此set不包含元素(空集)，此方法返回true。 |
| 6 | Iterator<E> iterator() 此方法返回一个迭代器在此set的元素。 |
| 7 | boolean remove(Object o) 此方法删除指定的元素，从这组(如果存在)。 |
| 8 | int size() 此方法返回返回元素在此set数字(它的基数)。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractSet
- java.util.AbstractCollection
- java.util.Object
- java.util.Set

# java.util.Hashtable类 - Java.util包

java.util.Hashtable 此类实现一个哈希表，该键映射到相应的值。以下是关于Hashtable的要点：

- 在此任何非空的对象可以被用作键或值。
- 如果很多条目将被制作成一个Hashtable，具有足够大的容量创建它可允许条目被插入的效率比让作为生长所需的表它执行自动再散列。

## 类声明

以下是java.util.Hashtable类的声明：

```
public class Hashtable<K,V>
   extends Dictionary<K,V>
      implements Map<K,V>, Cloneable, Serializable
```

## 类的构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | Hashtable() 这种构造一个新的空哈希表用默认的初始容量(11)和加载因子(0.75)。 |
| 2 | Hashtable(int initialCapacity) 这种构造一个新的空哈希表使用指定的初始容量和默认加载因子(0.75)。 |
| 3 | Hashtable(int initialCapacity, float loadFactor) 这种构造一个新的空哈希表与指定的初始容量和指定的加载因子。 |
| 4 | Hashtable(Map<? extends K,? extends V> t) 这种构造一个新的哈希表具有相同的映射给定的Map。 |

## 类方法

| S.N. | 方法与说明 |
|---|---|
| 1 | void clear() 此方法清除这个哈希表，以便它不包含任何键。 |
| 2 | Object clone() 此方法创建此哈希表的浅表副本。 |
| 3 | boolean contains(Object value) 此方法测试，如果一些键映射到该散列表中指定的值。 |
| 4 | boolean containsKey(Object key) 如果指定的对象是该散列表中的一个关键此方法测试。 |
| 5 | boolean containsValue(Object value) 如果此哈希表的一个或多个键映射到这个值此方法返回true。 |
| 6 | Enumeration<V> elements() 此方法在该散列表中返回值的枚举。 |
| 7 | Set<Map.Entry<K,V>> entrySet() 此方法返回此映射中包含的映射关系的Set视图。 |
| 8 | boolean equals(Object o) 此方法将指定对象与此映射的相等性比较，按照Map接口的定义。 |
| 9 | V get(Object key) 此方法返回指定键所映射的值，或者null，如果此映射不包含该键。 |
| 10 | int hashCode() 此方法返回按在Map接口的定义此Map的哈希码值。 |
| 11 | boolean isEmpty() 这如果此哈希表没有映射到按键值的方法测试。 |
| 12 | Enumeration<K> keys() 此方法返回此哈希表中的键的枚举。 |
| 13 | Set<K> keySet() 此方法返回此映射中包含的键的Set视图。 |
| 14 | V put(K key, V value) 这种方法映射到指定键在此哈希表中指定的值。 |
| 15 | void putAll(Map<? extends K,? extends V> t) 这个方法会将所有从指定映射到这个哈希表的映射。 |
| 16 | protected void rehash() 这种方法增加的容量和内部对其进行重组这个哈希表，以适应和更有效地访问其条目。 |
| 17 | V remove(Object key) 这个方法从哈希表中删除键(及其相应的值)。 |
| 18 | int size() 此方法返回此哈希表中的键的数量。 |
| 19 | String toString() 这种方法在一组条目的形式返回此Hashtable对象的字符串表示形式，括在括号由ASCII字符“,”(逗号加空格)分隔。 |
| 20 | Collection<V> values() 此方法返回此映射中包含的值的Collection视图。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.IdentityHashMap类 - Java.util包

**java.util.IdentityHashMap** 类实现了Map接口的哈希表，比较键(和值)时使用引用相等性代替对象相等的下面是IdentityHashMap有关的要点：

- 这个类提供了所有可选的映射操作，并且允许null值和null键。
- 这个类不保证为向地图的顺序;特别是，它不保证该顺序将继续随时间恒定。
- 在IdentityHashMap中，两个键k1和k2被认为是当且仅当(K1== K2)相等，而在Map实现(如HashMap的)两个键k1和k2被认为是相等当且仅当(K1== NULL？K2== NULL：k1.equals(K2))。

## 类声明

以下是java.util.IdentityHashMap类的声明：

```
public class IdentityHashMap<K,V>
   extends AbstractMap<K,V>
       implements Map<K,V>, Serializable, Cloneable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **IdentityHashMap()** 此构造一个新的空标识哈希映射具有默认期望最大尺寸(21)。 |
| 2 | **IdentityHashMap(int expectedMaxSize)** 此构造一个新的空地图与指定期望的最大尺寸。 |
| 3 | **IdentityHashMap(Map<? extends K,? extends V> m)** 这个构造包含指定映射中键 - 值映射关系的新标识哈希映射。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | void clear() 此方法删除所有来自此映射中的映射。 |
| 2 | Object clone() 此方法返回此标识哈希映射的浅表副本：键和值本身不被复制。 |
| 3 | boolean containsKey(Object key) 此方法测试指定的对象引用是否为此标识哈希映射中的键。 |
| 4 | boolean containsValue(Object value) 此方法测试指定的对象引用是否为值在此标识哈希映射。 |
| 5 | Set<Map.Entry<K,V>> entrySet() 此方法返回此映射中包含的映射关系的Set视图。 |
| 6 | boolean equals(Object o) 此方法将指定对象与此映射的相等性比较。 |
| 7 | V get(Object key) 这个方法返回指定键所映射的值，或者null，如果此映射不包含该键。 |
| 8 | int hashCode() 此方法返回返回这个映射的哈希码值。 |
| 9 | boolean isEmpty() 如果此标识哈希映射不包含键 - 值映射关系，此方法返回true。 |
| 10 | Set<K>keySet() 此方法返回此映射中包含的键的基于标识的set视图。 |
| 11 | V put(K key, V value) 此方法将指定的值与此标识哈希映射中的指定键。 |
| 12 | void putAll(Map<? extends K,? extends V> m) 这个方法会将所有从指定映射此映射中的映射。 |
| 13 | V remove(Object key) 此方法删除映射为这个键从该映射如果存在的话。 |
| 14 | int size() 此方法返回键 - 值映射关系在此标识哈希映射的数量。 |
| 15 | Collection<V> values() 此方法返回返回此映射中包含的值的Collection视图。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractMap
- java.util.Object

# java.util.LinkedHashMap类 - Java.util包

**java.util.LinkedHashMap** 类是哈希表和链接列表实现Map接口，具有可预知的迭代顺序。以下是关于LinkedHashMap的要点：

- 这个类提供了所有可选的映射操作，并且允许null元素。
- 在一个HashMap的迭代很可能会更加昂贵。

## 类声明

以下是java.util.LinkedHashMap类的声明：

```
public class LinkedHashMap<K,V>
   extends HashMap<K,V>
      implements Map<K,V>
```

## 参数

以下是java.util.LinkedHashMap类中的参数：

- K -- 这是映射要维护的键的类型。
- V -- 这是映射值的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **LinkedHashMap()** 这种构造具有默认初始容量(16)和加载因子(0.75)的空插入顺序LinkedHashMap实例。 |
| 2 | **LinkedHashMap(int initialCapacity)** 该构造带指定初始容量和默认加载因子(0.75)的空插入顺序LinkedHashMap实例。 |
| 3 | **LinkedHashMap(int initialCapacity, float loadFactor)** 该构造带指定初始容量和加载因子的空插入顺序LinkedHashMap实例。 |
| 4 | **LinkedHashMap(int initialCapacity, float loadFactor, boolean accessOrder)** 该构造带指定初始容量，加载因子和排序模式的空LinkedHashMap实例。 |
| 5 | **LinkedHashMap(Map<? extends K,? extends V> m)** 这与构建相同的映射关系与指定映射的插入顺序LinkedHashMap实例。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void clear() 此方法删除所有来自此映射中的映射。 |
| 2 | boolean containsValue(Object value) 如果此映射的一个或多个键映射到指定值，该方法返回true。 |
| 3 | V get(Object key) 这个方法返回指定键所映射的值，或者null，如果此映射不包含该键。 |
| 4 | protected boolean removeEldestEntry(Map.Entry<K,V> eldest) 这个方法如果此映射移除其最旧的条目返回true。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.HashMap
- java.util.AbstarctMap
- java.util.Object
- java.util.Map

# java.util.LinkedHashSet类 - Java.util包

**java.util.LinkedHashSet** 类是一个哈希表和链接列表实现Set接口，具有可预知的迭代顺序。以下是关于LinkedHashSet的要点：

- 这个类提供了所有可选set操作，并且允许null元素。

## 类声明

以下是java.util.LinkedHashSet类的声明：

```
public class LinkedHashSet<E>
   extends HashSet<E>
     implements Set<E>, Cloneable, Serializable
```

## 参数

以下是java.util.LinkedHashSet类中的参数：

- E -- 这是此set维护元素的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | LinkedHashSet() 这种构造一个新的空链接哈希集合具有默认初始容量（16）和加载因子（0.75）。 |
| 2 | LinkedHashSet(Collection<? extends E> c) 这构造了一个新的链接哈希集合具有相同元素的指定集合。 |
| 3 | LinkedHashSet(int initialCapacity) 这种构造一个新的空链接哈希集合具有指定初始容量和默认加载因子（0.75）。 |
| 4 | LinkedHashSet(int initialCapacity, float loadFactor) 这种构造一个新的空链接哈希集合具有指定初始容量和加载因子。 |

## 类方法

这个类从以下类继承的方法：

- java.util.HashSet

- java.util.AbstractSet
- java.util.AbstractCollection
- java.util.Object
- java.util.Set

# java.util.LinkedList类 - Java.util包

**java.util.LinkedList** 类操作执行，我们可以预期这是一个双向链表。操作的索引列表会遍历从一开始或结束时，取其靠近指定索引的列表。

## 类声明

以下是java.util.LinkedList类的声明：

```
public class LinkedList<E>
   extends AbstractSequentialList<E>
      implements List<E>, Deque<E>, Cloneable, Serializable
```

## 参数

以下是java.util.LinkedList类的参数：

- E -- 这是在这个集合所持元素的类型。

## 字段域

从java.util.AbstractLis类t的字段继承。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | LinkedList() 这种构造构造一个空的列表。 |
| 2 | LinkedList(Collection<? extends E> c) 这个构造一个包含指定集合中的元素的列表，它们被集合的迭代器返回的顺序。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean add(E e) 这种方法将指定元素追加到此列表的末尾。 |
| 2 | void add(int index, E element) 此方法将在此列表中指定位置的指定元素。 |

| 3 | boolean addAll(Collection<? extends E> c) 此方法会将所有指定集合中的元素添加到此列表的结尾，因为它们是由指定collection的迭代器返回的顺序。 |
|---|---|
| 4 | boolean addAll(int index, Collection<? extends E> c) 此方法将所有指定集合中的元素插入此列表，开始在指定的位置。 |
| 5 | void addFirst(E e) 此方法返回插入指定的元素，在这个列表的开头.. |
| 6 | void addLast(E e) 此方法返回指定元素追加到此列表的末尾。 |
| 7 | void clear() 此方法删除所有来自此列表中的元素。 |
| 8 | Object clone() 此方法返回返回此LinkedList的浅表副本。 |
| 9 | boolean contains(Object o) 如果此列表包含指定的元素，此方法返回true。 |
| 10 | Iterator<E> descendingIterator() 此方法返回一个迭代器在此双端队列以逆向顺序的元素。 |
| 11 | E element() 此方法检索，但是不移除此列表的头（第一个元素）。 |
| 12 | E get(int index) 此方法返回的元素在此列表中的指定位置。 |
| 13 | E getFirst() 此方法返回此列表中的第一个元素。 |
| 14 | E getLast() 此方法返回此列表中的最后一个元素。 |
| 15 | int indexOf(Object o) 这个方法返回指定元素的第一个匹配项的索引在此列表中，或者-1，如果此列表中不包含该元素。 |
| 16 | int lastIndexOf(Object o) 这个方法返回指定元素的最后一个匹配项的索引在此列表中，或者-1，如果此列表中不包含该元素。 |
| 17 | ListIterator<E> listIterator(int index) 此方法返回一个列表迭代器在此列表中的元素（按适当顺序），从列表中的指定位置。 |
| 18 | boolean offer(E e) 此方法将指定的元素到此列表的末尾（最后一个元素）。 |
| 19 | boolean offerFirst(E e) 此方法将指定元素在此列表的前面。 |
| 20 | boolean offerLast(E e) 此方法将指定的元素，在这个列表的末尾。 |
| 21 | E peek() 此方法检索，但是不移除此列表的头（第一个元素）。 |
| 22 | E peekFirst() 此方法检索，但是不移除此列表的第一个元素，或者如果此列表为空，则返回null。 |
| 23 | E peekLast() 此方法检索，但是不移除此列表的最后一个元素，或者如果此列表为空，则返回null。 |
| 24 | E poll() 此方法检索并移除此列表的头（第一个元素）。 |
| 26 | E pollFirst() 此方法检索并移除此列表的第一个元素，或者如果此列表为 |

| | |
|---|---|
| 26 | 空，则返回null。 |
| 27 | E pollLast() 此方法检索并移除此列表的最后一个元素，或者如果此列表为空，则返回null。 |
| 28 | E pop() 这个方法从该列表所表示的堆栈弹出一个元素。 |
| 29 | void push(E e) 这种方法将元素推入此列表所表示的堆栈。 |
| 30 | E remove() 此方法检索并移除此列表的头（第一个元素）。 |
| 31 | E remove(int index) 此方法删除的元素在此列表中指定位置. |
| 32 | boolean remove(Object o) 此方法从该列表中首次出现的指定元素，如果它存在。 |
| 33 | E removeFirst() 此方法删除并返回此列表的第一个元素。 |
| 34 | boolean removeFirstOccurrence(Object o) 此方法删除指定元素第一次出现在该列表中（遍历从头部到尾部列表时）。 |
| 35 | E removeLast() 此方法删除并返回此列表的最后一个元素。 |
| 36 | boolean removeLastOccurrence(Object o) 此方法删除指定元素最后一次出现在该列表中（遍历从头部到尾部列表时）。 |
| 37 | E set(int index, E element) 这种方法取代在与指定的元素在此列表中指定位置的元素。 |
| 38 | int size() 此方法返回此列表中的元素数。 |
| 39 | Object[] toArray() 这个方法返回一个包含所有在此列表中正确的序列中元素的数组（从第一个到最后一个元素）。 |
| 40 | <T> T[] toArray(T[] a) 这个方法返回一个包含所有在此列表中正确的序列中的元素（从第一个到最后一个元素）一个数组，返回数组的运行时类型是指定数组的。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractSequentialList
- java.util.AbstractList
- java.util.AbstractCollection
- java.util.Object
- java.util.List
- java.util.Deque

# java.util.ListResourceBundle类 - Java.util包

java.util.ListResourceBundle 类是资源包的抽象子类，在一个方便和易于使用列表管理资源语言环境。

## 类声明

以下是java.util.ListResourceBundle类的声明：

```
public abstract class ListResourceBundle
    extends ResourceBundle
```

## 字段域

字段从 java.util.ResourceBundle 类继承

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **ListResourceBundle()** 这是唯一的构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | protected abstract Object[][] getContents() 此方法返回一个数组，其中每个项目是一个对对象的Object数组。 |
| 2 | Enumeration<String> getKeys() 此方法返回包含在此的ResourceBundle及其父包中的键的枚举。 |
| 3 | Object handleGetObject(String key) 这种方法获取的对象从这个资源包的给定键。 |
| 4 | protected Set<String> handleKeySet() 此方法返回一个Set仅包含在此ResourceBundle中的键。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.ResourceBundle
- java.util.Object

# java.util.Locale类 - Java.util包

**java.util.Locale**类对象表示了特定的地理，政治和文化地区。 以下是有关区域设置的要点：

- 需要Locale来执行其任务的操作称为语言环境敏感，它使用Locale，形成信息的用户。
- Locale是一种机制，用于识别对象，而不是一个容器对象本身。

## 类声明

以下是java.util.Locale类的声明：

```
public final class Locale
   extends Object
      implements Cloneable, Serializable
```

## 字段域

以下是java.util.Locale类中的字段：

- static Locale CANADA -- 这是国家的常量。
- static Locale CANADA FRENCH -- 这是国家的常量。
- static Locale CHINA -- 这是国家的常量。
- static Locale CHINESE -- 这是语言的常量。
- static Locale ENGLISH -- 这是语言的常量。
- static Locale FRANCE -- 这是国家的常量。
- static Locale FRENCH -- 这是语言的常量。
- static Locale GERMAN -- 这是语言的常量。
- static Locale GERMANY -- 这是国家的常量。
- static Locale ITALIAN -- 这是语言的常量。
- static Locale ITALY -- 这是国家的常量。
- static Locale JAPAN -- 这是国家的常量。
- static Locale JAPANESE -- 这是语言的常量。

- static Locale KOREA -- 这是国家的常量。
- static Locale KOREAN -- 这是语言的常量。
- static Locale PRC -- 这是国家的常量。
- static Locale ROOT -- 这是不变的根语言环境。
- static Locale SIMPLIFIED CHINESE -- 这是语言的常量。
- static Locale TAIWAN -- 这是国家的常量。
- static Locale TRADITIONAL CHINESE -- 这是语言的常量。
- static Locale UK -- 这是国家的常量。
- static Locale US -- 这是国家的常量。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | Locale(String language) 这个构造一个语言环境的语言代码。 |
| 2 | Locale(String language, String country) 这个构造一个语言环境的语言代码。 |
| 3 | Locale(String language, String country, String variant) 这个构造一个语言环境的语言，国家，变体。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Object clone() 此方法重写了Cloneable |
| 2 | boolean equals(Object obj) 如果此区域设置为等于另一个对象，则此方法返回true。 |
| 3 | static Locale[] getAvailableLocales() 此方法返回所有已安装的语言环境的数组。 |
| 4 | String getCountry() 此方法返回国家/地区代码为这个区域设置，它要么是空字符串或大写的ISO3166两字母代码。 |
| 5 | static Locale getDefault() 此方法获取默认语言环境的当前值的Java虚拟机实例。 |
| 6 | String getDisplayCountry() 此方法返回适合显示给用户的名称语言环境的国家。 |

| | |
|---|---|
| 7 | S String getDisplayCountry(Locale inLocale) 此方法返回适合显示给用户的名称语言环境的国家。 |
| 8 | String getDisplayLanguage() 此方法返回的语言环境语言适合于显示给用户的名称。 |
| 9 | String getDisplayLanguage(Locale inLocale) 此方法返回的语言环境语言适合于显示给用户的名称。 |
| 10 | String getDisplayName() 此方法返回的语言环境，是否适合显示给用户的名称。 |
| 11 | String getDisplayName(Locale inLocale) 此方法返回的语言环境，是否适合显示给用户的名称。 |
| 12 | String getDisplayVariant() 此方法返回的语言环境变量代码，适合显示给用户的名称。 |
| 13 | String getDisplayVariant(Locale inLocale) 此方法返回的语言环境变量代码，适合显示给用户的名称。 |
| 14 | String getISO3Country() 此方法返回一个三个字母的缩写本地区的国家。 |
| 15 | String getISO3Language() 此方法返回返回一个三字母缩写在这个地方的语言。 |
| 16 | static String[] getISOCountries() 这个方法返回ISO 3166中定义的所有2个字母的国家代码的列表。 |
| 17 | static String[] getISOLanguages() 这个方法返回ISO 639中定义的所有两字母语言代码的列表。 |
| 18 | String getLanguage() 此方法返回语言代码为这个区域设置，它要么是空字符串或小写的ISO639代码。 |
| 19 | String getVariant() 此方法返回的变量代码为这个区域设置。 |
| 20 | int hashCode() 这个方法重写了hashCode。 |
| 21 | static void setDefault(Locale newLocale) 此方法设置的默认语言环境的Java虚拟机实例。 |
| 22 | String toString() 这种方法是的getter整个语言环境的编程名称，由下划线分隔的语言，国家和变量。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.Observable类 - Java.util包

**java.util.Observable** 类表示Observable的对象，或在模型视图范例“data”。以下是关于Observable的要点：

- 这个类可以被子类化表示对象的应用程序要观察。
- observable的对象可以具有一个或多个observers。

## 类声明

以下是java.util.Observable类的声明：

```
public class Observable
   extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Observable()** 这构造了一个Observable的带有零个Observers。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void addObserver(Observer o) 这种方法增加了observer所设定的observer对象，只要它是不同于一些observer已经在此集合。 |
| 2 | protected void clearChanged() 此方法指示此对象不再改变，或者说，它已经通知其所有的最新变化及其observers，所以hasChanged方法现在将返回false。 |
| 3 | int countObservers() 这个方法返回当前Observable对象的observers数量。 |
| 4 | void deleteObserver(Observer o) 此方法从该集合对象的observers删除一个observer。 |
| 5 | void deleteObservers() 此方法清除observers 列表，使此对象不再有任何observer 。 |
| 6 | boolean hasChanged() 此方法测试，如果该对象已经改变。 |
| 7 | void notifyObservers() 如果该对象已经改变，由hasChanged方法指示，则通知其所有观察者，并调用clearChanged方法来指示此对象不再改变。 |
| 8 | void notifyObservers(Object arg) 如果该对象已经改变，由hasChanged方法指示，则通知其所有观察者，并调用clearChanged方法来指示此对象不再改变。 |
| 9 | protected void setChanged() 此方法返回这个标记Observable对象为已改变; hasChanged方法现在将返回true。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.PriorityQueue类 - Java.util包

**java.util.PriorityQueue** 类是基于优先级堆的无界优先级队列。以下是有关的PriorityQueue的要点：

- 优先级队列中的元素根据其自然顺序进行排序，或者按照队列构造时提供的Comparator，这取决于使用的构造方法。
- 优先级队列不允许null元素。
- 优先级队列依靠自然顺序也不允许插入不可比较的对象。

## 类声明

以下是java.util.PriorityQueue类的声明：

```
public class PriorityQueue<E>
   extends AbstractQueue<E>
      implements Serializable
```

## 参数

以下是java.util.PriorityQueue类的参数：

- E -- 这是在这个集合所持元素的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **PriorityQueue()** 这将创建一个具有默认初始容量(11)，根据其自然顺序对其元素PriorityQueue。 |
| 2 | **PriorityQueue(Collection<? extends E> c)** 这将创建一个包含指定集合中的元素的PriorityQueue。 |
| 3 | **PriorityQueue(int initialCapacity)** 这将创建与根据其自然顺序对其元素指定的初始容量创建一个PriorityQueue。 |
| 4 | **PriorityQueue(int initialCapacity, Comparator<? super E> comparator)** 这将创建与根据指定的比较器对其元素指定的初始容量创建一个PriorityQueue。 |
| 5 | **PriorityQueue(PriorityQueue<? extends E> c)** 这将创建一个包含指定优先级队列中的元素一个PriorityQueue。 |
| 6 | **PriorityQueue(SortedSet<? extends E> c)** 这将创建一个包含指定有序set的元素一个PriorityQueue。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean add(E e) 此方法将指定元素插入此优先级队列。 |
| 2 | void clear() 此方法删除所有来自此优先级队列中的元素。 |
| 3 | Comparator<? super E> comparator() 此方法返回用于排序在此队列中，或者为null的元素;如果此队列根据其元素的自然顺序进行排序的比较器。 |
| 4 | boolean contains(Object o) 如果此队列包含指定的元素，此方法返回true。 |
| 5 | Iterator<E> iterator() 此方法返回一个迭代器在此队列中的元素。 |
| 6 | boolean offer(E e) 此方法将指定元素插入此优先级队列。 |
| 7 | E peek() 此方法检索，但是不移除此队列的头，如果此队列为空，则返回null。 |
| 8 | E poll() 此方法检索并移除此队列的头，如果此队列为空，则返回null。 |
| 9 | boolean remove(Object o) 这个方法从队列中移除指定元素的单个实例(如果存在)。 |
| 10 | int size() 此方法返回这个集合中元素的个数。 |
| 11 | Object[] toArray() 这个方法返回一个包含此队列所有元素的数组。 |
| 12 | <T> T[] toArray(T[] a) 这个方法返回一个包含此队列所有元素的数组;返回数组的运行时类型是指定数组。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.AbstractQueue
- java.util.AbstractCollection
- java.util.Object
- java.util.Collection

# java.util.Properties类 - Java.util包

**java.util.Properties**类是代表一个持久的一套详细属性，属性可以被保存到一个流或从流中加载的类。以下是关于属性的要点：

- 属性列表中每个键及其对应值是一个字符串。
- 一个属性列表可包含另一个属性列表作为它的“默认”，第二个属性可在列表中搜索，如果没有在原有的属性列表中找到的属性键。
- 这个类是线程安全的;多个线程可以共享一个Properties对象，而不需要外部同步。

## 类的声明

以下是java.util.Properties类的声明：

```
public class Properties
    extends Hashtable<Object,Object>
```

## 字段域

下面是一个java.util.Properties类中的字段：

- protected Properties defaults -- 这是包含在属性列表中未找到任何键的默认值的属性列表。

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **Properties()** 这种构造创建一个空的属性列表，没有默认值。 |
| 2 | **Properties(Properties defaults)** 这种构造创建一个空的属性列表中具有指定默认值。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | String getProperty(String key) 该方法将搜索具有此属性列表中指定键的属性。 |
| 2 | String getProperty(String key, String defaultValue) 该方法将搜索具有此属性列表中指定键的属性。 |
| 3 | void list(PrintStream out) 这种方法打印属性列表输出到指定的输出流。 |
| 4 | void list(PrintWriter out) 这种方法打印属性列表输出到指定的输出流。 |
| 5 | void load(InputStream inStream) 此方法读取属性列表(键和元素对)从输入字节流。 |
| 6 | void load(Reader reader) 这个方法从一个简单的面向行的格式输入字符流中读取属性列表(键和元素对)。 |
| 7 | void loadFromXML(InputStream in) 此方法加载所有指定的输入流中到此属性表中的XML文档所表示的所有属性。 |
| 8 | Enumeration<?> propertyNames() 此方法返回属性列表中所有键，包括默认属性列表中不同的键的枚举，如果尚未发现从主属性列表中名称相同的键。 |
| 9 | void save(OutputStream out, String comments) 此方法读取a |
| 10 | Object setProperty(String key, String value) 此方法调用Hashtable的put()方法。 |
| 11 | void store(OutputStream out, String comments) 该方法写入此属性列表(键和元素对)在此属性表中适于装成一个属性表中使用load(InputStream)方法的格式输出流。 |
| 10 | void store(Writer writer, String comments) 该方法写入此属性列表(键和元素对)在此属性表中适合使用load(Reader)方法的格式输出字符流。 |
| 11 | void storeToXML(OutputStream os, String comment) 这个方法会发出代表所有包含在此表中的属性的XML文档。 |
| 12 | void storeToXML(OutputStream os, String comment, String encoding) 这个方法会发出代表所有包含在此表中的属性的XML文档，使用指定的编码。 |
| 13 | Set<String> stringPropertyNames() 此方法返回一组键在此属性列表，其中的键及其对应值是字符串，包括默认属性列表中不同的键，如果尚未发现从主属性列表中同名的键。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Hashtable
- java.util.Object

# java.util.PropertyPermission类 - Java.util包

**java.util.PropertyPermission**类是一个类属性的权限。以下是关于PropertyPermission要点：

- 该名称是属性的名称 ("java.home", "os.name", etc).
- 命名约定遵守层次结构属性命名约定。星号可以出现在名称的结尾，有一个“.”，或者本身以表示通配符。例如： "java.*" *or* "*" 是有效的, 而 "*java" 或"a*b" 是无效的.

## 类声明

以下是java.util.PropertyPermission类的声明：

```
public final class PropertyPermission
   extends BasicPermission
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **PropertyPermission(String name, String actions)** 这将创建具有指定名称的新PropertyPermission对象。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean equals(Object obj) 此方法检查相等的两个PropertyPermission对象。 |
| 2 | String getActions() 此方法返回动作的“规范化字符串表示形式”。 |
| 3 | int hashCode() 此方法返回此对象的哈希码值。 |
| 4 | boolean implies(Permission p) 此方法检查，如果这PropertyPermission对象是否“暗含”指定的权限。 |
| 5 | PermissionCollection newPermissionCollection() 此方法返回一个用于存储PropertyPermission对象返回一个新的PermissionCollection对象。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Permission
- java.util.Object

# java.util.PropertyResourceBundle类 - Java.util包

**java.util.PropertyResourceBundle** 类是资源包的具体子类使用一组从一个属性文件的静态字符串，用于管理一个本地化的资源。以下是有关的PropertyResourceBundle要点：

- 这个类可以从InputStream或Reader，该代表一个属性文件来构建。

## 类声明

以下是java.util.PropertyResourceBundle类的声明：

```
public class PropertyResourceBundle
   extends ResourceBundle
```

## 字段域

字段从类java.util.ResourceBundle 继承

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **PropertyResourceBundle(InputStream stream)** 这将从InputStream属性创建资源包。 |
| 2 | **PropertyResourceBundle(Reader reader)** 这将从读取器的属性创建资源包。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Enumeration<String> getKeys() 此方法返回包含在此ResourceBundle及其父包中的键的枚举。 |
| 2 | Object handleGetObject(String key) 这个方法从这个资源包的给定键获取对象。 |
| 3 | protected Set<String> handleKeySet() 此方法返回一个Set仅包含在此ResourceBundle中的键。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.ResourceBundle
- java.util.Object

# java.util.Random类 - Java.util包

**java.util.Random** 类实例用于生成伪随机数。下面是有关Random要点：

- 这个类使用一个48位的种子，这是一个线性同余公式修改。
- 该算法通过**Random**类的使用在每次调用时可以提供高达32生成的伪随机位受保护的实用方法来实现。

## 类声明

以下是java.util.Random类的声明：

```
public class Random
   extends Object
      implements Serializable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **Random()** 这将创建一个新的随机数生成器。 |
| 2 | **Random(long seed)** 这会使用单个long种子一个新的随机数发生器。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | protected int next(int bits) 此方法生成下一个伪随机数。 |
| 2 | boolean nextBoolean() 此方法返回下一个伪从这个随机数生成器的序列中均匀分布的boolean值。 |
| 3 | void nextBytes(byte[] bytes) 此方法生成随机字节并将其置于用户提供的字节数组。 |
| 4 | double nextDouble() 此方法返回下一个伪从这个随机数生成器的序列中均匀分布的0.0和1.0之间的double值。 |
| 5 | float nextFloat() 此方法返回下一个伪从这个随机数生成器的序列中均匀分布的0.0和1.0之间的float值。 |
| 6 | double nextGaussian() 此方法返回下一个伪高斯(“正常地”)分布的均值为0.0，标准差为1.0从此随机数生成器的序列的double值。 |
| 7 | int nextInt() 此方法返回下一个伪从这个随机数生成器的序列中均匀分布的int值。 |
| 8 | int nextInt(int n) 此方法返回一个伪随机，均匀分布在0(含)int值和指定值(不包括)，从此随机数生成器的序列中取出的。 |
| 9 | long nextLong() 此方法返回下一个伪从这个随机数生成器的序列中均匀分布的long值。 |
| 10 | void setSeed(long seed) 此方法设置此随机数生成器的使用单个long种子的种子。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.ResourceBundle类 - Java.util包

**java.util.ResourceBundle** 类包含特定于语言环境的对象。下面是有关资源包的要点：

- 该类允许您轻松编写地本地化或翻译成不同语言开发的程序。
- 该类程序处理多个语言环境以后可以轻松进行修改，以支持更多的语言环境。
- Java平台提供ResourceBundle，ListResourceBundle和PropertyResourceBundle两个子类。

## 类声明

以下是java.util.ResourceBundle类的声明：

```
public abstract class ResourceBundle
   extends Object
```

## 字段域

以下是java.util.ResourceBundle类中的字段：

- protected ResourceBundle parent -- 这是此包的父包。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **ResourceBundle()** 这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | static void clearCache() 此方法从已经使用调用者的类加载器加载的缓存中的所有资源包。 |
| 2 | static void clearCache(ClassLoader loader) 此方法从已经使用给定的类加载器加载的缓存中移除所有资源包。 |
| 3 | boolean containsKey(String key) 此方法确定给定键是否包含在这个资 |

| | |
|---|---|
| 3 | 源包或它的父包。 |
| 4 | static ResourceBundle getBundle(String baseName) 此方法使用指定的基本名称，默认的语言环境和调用者的类加载器获取资源包。 |
| 5 | static ResourceBundle getBundle(String baseName, Locale locale) 此方法使用指定的基本名称和语言环境和调用者的类加载器获取资源包。 |
| 6 | static ResourceBundle getBundle(String baseName, Locale locale, ClassLoader loader) 此方法使用指定的基本名称，语言环境和类加载器获取资源包。 |
| 7 | static ResourceBundle getBundle(String baseName, Locale targetLocale, ClassLoader loader, ResourceBundle.Control control) 此方法返回一个使用指定基本名称，目标语言环境，类加载器和控件返回资源包。 |
| 8 | static ResourceBundle getBundle(String baseName, Locale targetLocale, ResourceBundle.Control control) 此方法返回使用指定的基本名称，目标语言环境和控制，以及调用者的类加载器获取资源包。 |
| 9 | static ResourceBundle getBundle(String baseName, ResourceBundle.Control control) 此方法返回使用指定的基本名称，默认的语言环境和指定控件的资源包。 |
| 10 | abstract Enumeration<String> getKeys() 此方法返回键的枚举。 |
| 11 | Locale getLocale() 此方法返回此资源包的语言环境。 |
| 12 | Object getObject(String key) 此方法获取一个对象给定键从此资源包或它的某个父类。 |
| 13 | String getString(String key) 此方法获取一个字符串给定键从此资源包或它的某个父类。 |
| 14 | String[] getStringArray(String key) 此方法得到的字符串数组给定键从此资源包或它的某个父类。 |
| 15 | protected abstract Object handleGetObject(String key) 此方法获取的对象从这个资源包的给定键。 |
| 16 | protected Set<String> handleKeySet() 此方法查询，如果给定的日期是在夏令时在该时区。 |
| 17 | Set<String> keySet() 此方法返回一个Set包含在这个资源包及其父包中的所有键。 |
| 18 | protected void setParent(ResourceBundle parent) 此方法设置此包的父包。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.ResourceBundle.Control类 - Java.util包

**java.util.ResourceBundle.Control** 类具有协作的工厂方法用于加载资源包。以下是有关ResourceBundle.Control要点：

- 类实例必须是线程安全的，如果它同时使用多个线程。

## 类声明

以下是java.util.ResourceBundle.Control类的声明：

```
public static class ResourceBundle.Control
   extends Object
```

## 字段域

以下是java.util.ResourceBundle.Control类中的字段：

- static int STANDARD_TIME -- 这是常量的开始或结束时间的规定为标准时间的模式。
- static List<String> FORMAT_CLASS -- 这是一个包含“java.class”的格式列表。
- static List<String> FORMAT_DEFAULT -- 这是默认的格式列表，其中包含字符串“的java.class”和“java.properties”，在这个秩序。
- static List<String> FORMAT_PROPERTIES -- 这是一个包含“java.properties”的属性，唯一格式列表。
- static long TTL_DONT_CACHE -- 这是时间的生存常数不缓存加载资源包实例。
- static long TTL_NO_EXPIRATION_CONTROL -- 这是时间的生存常数禁用到期控制在缓存中加载资源包实例。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **List<Locale> getCandidateLocales(String baseName, Locale locale)** 这是一个构造函数。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | List<Locale>getCandidateLocales(String baseName, Locale locale) 此方法返回的区域设置列表作为候选语言环境的baseName和语言环境。 |
| 2 | static ResourceBundle.Control getControl(List<String> formats) 此方法返回一个的ResourceBundle.Control其中getFormats方法返回指定格式。 |
| 3 | Locale getFallbackLocale(String baseName, Locale locale) 此方法返回由ResourceBundle.getBundle工厂方法被用来作为一个备用的区域进行进一步资源包搜索一个区域设置。 |
| 4 | List<String> getFormats(String baseName) 这个方法返回一个包含格式可以用来加载资源包为给定baseName字符串列表。 |
| 5 | static ResourceBundle.Control getNoFallbackControl(List<String> formats) 此方法返回一个ResourceBundle.Control，其中getFormats方法返回指定的格式和getFallbackLocale方法返回null。 |
| 6 | long getTimeToLive(String baseName, Locale locale) 此方法返回的时间到现场（TTL）值此项下ResourceBundle.Control加载资源包。 |
| 7 | boolean needsReload(String baseName, Locale locale, String format, ClassLoader loader, ResourceBundle bundle, long loadTime) 该方法确定是否需要在高速缓存中过期束根据loadTime或一些其它标准给出的加载时间来重新加载。 |
| 8 | ResourceBundle newBundle(String baseName, Locale locale, String format, ClassLoader loader, boolean reload) 此方法实例化给定的格式和语言环境的给定包名称的资源包，使用给定的类加载器，如果必要的。 |
| 9 | String toBundleName(String baseName, Locale locale) 该方法给出baseName和语言环境的包名称转换。 |
| 10 | String toResourceName(String bundleName, String suffix) 这种方法给定的bundleName的转换为通过替换所有出现所需的ClassLoader.getResource法的形式'.'在bundleName以'/'，并附加一个'.'和给定的文件后缀。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.Scanner类 - Java.util包

java.util.Scanner 类是一个简单的文本扫描器可以分析基本类型和字符串使用正则表达式。以下是关于扫描器的要点：

- 一个扫描器使用分隔符模式分解它的输入，默认情况下与空白匹配。
- 扫描操作可能阻塞等待输入。
- 扫描器是不是安全的，无需外部同步多线程使用。

## 类声明

以下是java.util.Scanner类的声明：

```
public final class Scanner
   extends Object
      implements Iterator<String>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Scanner(File source)** 此构造产生从指定文件扫描值的新扫描器。 |
| 2 | **Scanner(File source, String charsetName)** 此构造产生从指定文件扫描值的新扫描器。 |
| 3 | **Scanner(InputStream source)** 此构造产生从指定的输入流扫描值的新扫描器。 |
| 4 | **Scanner(InputStream source, String charsetName)** 此构造产生从指定的输入流扫描值的新扫描器。 |
| 5 | **Scanner(Readable source)** 此构造产生从指定源扫描值的新扫描器。 |
| 6 | **Scanner(ReadableByteChannel source)** 此构造产生从指定通道扫描值的新扫描器。 |
| 7 | **Scanner(ReadableByteChannel source, String charsetName)** 此构造产生从指定通道扫描值的新扫描器。 |
| 8 | **Scanner(String source)** 此构造产生从指定字符串扫描值的新扫描器。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void close() 此方法关闭此扫描器。 |
| 2 | Pattern delimiter() 此方法返回当前扫描器使用相匹配的分隔符的模式。 |
| 3 | String findInLine(Pattern pattern) 此方法试图找到指定的模式在忽略分隔符的下一次出现。 |
| 4 | String findInLine(String pattern) 此方法试图找到从指定字符串构造的模式在未来发生，忽略分隔符。 |
| 5 | String findWithinHorizon(Pattern pattern, int horizon) 此方法试图找到指定模式的下一次出现。 |
| 6 | String findWithinHorizon(String pattern, int horizon) 此方法试图找到从指定字符串构造的模式在未来发生，在忽略分隔符。 |
| 7 | boolean hasNext() 如果此扫描器有另一个标记在其输入此方法返回true。 |
| 8 | boolean hasNext(Pattern pattern) 如果下一个完整标记与指定模式匹配，该方法返回true。 |
| 9 | boolean hasNext(String pattern) 如果下一个标记与从指定字符串构造的模式此方法返回true。 |
| 10 | boolean hasNextBigDecimal() 如果在此扫描器输入信息中的下一个标记可以通过使用nextBigDecimal()方法被解释为一个BigDecimal，该方法返回true。 |
| 11 | boolean hasNextBigInteger() 如果在此扫描器输入信息中的下一个标记可以使用nextBigInteger()方法被解释为一个BigInteger的默认基数，此方法返回true。 |
| 12 | boolean hasNextBigInteger(int radix) 如果在此扫描器输入信息中的下一个标记可以使用nextBigInteger()方法被解释为一个BigInteger指定基数此方法返回true。 |
| 13 | boolean hasNextBoolean() 如果在此扫描器输入信息中的下一个标记可以解释为使用从字符串“true\|false”创造了一个不区分大小写模式的布尔值，此方法返回true。 |
| 14 | boolean hasNextByte() 如果在此扫描器输入信息中的下一个标记可以使用nextByte()方法被解释为一个字节值的默认基数此方法返回true。 |
| 15 | boolean hasNextByte(int radix) 如果在此扫描器输入信息中的下一个标记可以使用nextByte()方法被解释为一个字节值指定基数，此方法返回true。 |
| 16 | boolean hasNextDouble() 如果在此扫描器输入信息中的下一个标记可以解释为使用nextDouble()方法的一个double值，此方法返回true。 |
|  | boolean hasNextFloat() 此方法返回true，如果在此扫描器输入信息的下 |

| | |
|---|---|
| | 一个标记可以使用nextFloat()方法解释为浮点值。 |
| 18 | boolean hasNextInt() 如果在此扫描器输入信息中的下一个标记可以使用nextInt()方法被解释为一个int值的默认基数，此方法返回true。 |
| 19 | boolean hasNextInt(int radix) 如果在此扫描器输入信息中的下一个标记可以使用nextInt()方法被解释为一个int值指定基数，此方法返回true。 |
| 20 | boolean hasNextLine() 如果有另一行在此扫描器的输入，此方法返回true。 |
| 21 | boolean hasNextLong() 如果在此扫描器输入信息中的下一个标记可以使用nextLong()方法被解释为一个long值的默认基数，此方法返回true。 |
| 22 | boolean hasNextLong(int radix) 如果在此扫描器输入信息中的下一个标记可以使用nextLong()方法被解释为一个long值指定基数，此方法返回true。 |
| 23 | boolean hasNextShort() 如果在此扫描器输入信息中的下一个标记可以使用nextShort()方法被解释为一个short值的默认基数，此方法返回true。 |
| 24 | boolean hasNextShort(int radix) 如果在此扫描器输入信息中的下一个标记可以使用nextShort()方法被解释为一个short值指定基数，此方法返回true。 |
| 25 | IOException ioException() 此方法返回的最后一个IOException异常抛出此扫描器的基本可读。 |
| 26 | Locale locale() 此方法返回此扫描器的语言环境。 |
| 27 | MatchResult match() 此方法返回此扫描器所执行的最后扫描操作的匹配结果。 |
| 28 | String next() 此方法查找并返回来自此扫描器的下一个完整标记。 |
| 29 | String next(Pattern pattern) 此方法返回下一个标记，如果它与指定模式匹配。 |
| 30 | String next(String pattern) 此方法返回下一个标记，如果它匹配从指定字符串构造的模式。 |
| 31 | BigDecimal nextBigDecimal() 此方法扫描输入的下一个标记为一个BigDecimal。 |
| 32 | BigInteger nextBigInteger() 此方法扫描输入的下一个标记为一个BigInteger。 |
| 33 | BigInteger nextBigInteger(int radix) 此方法扫描输入的下一个标记为一个BigInteger。 |
| 34 | boolean nextBoolean() 此方法扫描输入的下一个标记成一个布尔值并返回该值。 |
| 35 | byte nextByte() 此方法扫描输入的下一个标记为一个字节。 |

| | |
|---|---|
| 35 | byte nextByte() 此方法扫描输入的下一个标记为一个字节。 |
| 36 | byte nextByte(int radix) 此方法扫描输入的下一个标记为一个字节。 |
| 37 | double nextDouble() 此方法扫描输入的下一个标记为double。 |
| 38 | float nextFloat() 此方法扫描输入的下一个标记为float。 |
| 39 | int nextInt() 此方法扫描输入的下一个标记为int。 |
| 40 | int nextInt(int radix) 此方法扫描输入的下一个标记为int。 |
| 41 | String nextLine() 此方法前进此扫描器执行当前行，并返回跳过的输入信息。 |
| 42 | long nextLong() 此方法扫描输入的下一个标记为一个long。 |
| 43 | long nextLong(int radix) 此方法扫描输入的下一个标记为一个long。 |
| 44 | short nextShort() 此方法扫描输入的下一个标记为一个long。 |
| 45 | short nextShort(int radix) 此方法扫描输入的下一个标记为short。 |
| 46 | int radix() 此方法返回此扫描器的默认基数。 |
| 47 | void remove() 不受此实现迭代器所支持的删除操作。 |
| 48 | Scanner reset() 此方法重置该扫描仪。 |
| 49 | Scanner skip(Pattern pattern) 此方法跳过输入相匹配的指定模式，在忽略分隔符。 |
| 50 | Scanner skip(String pattern) 此方法跳过输入匹配从指定字符串构造的模式。 |
| 51 | String toString() 此方法返回当前扫描器的字符串表示形式。 |
| 52 | Scanner useDelimiter(Pattern pattern) 此方法设置此扫描器的分隔模式，以指定的模式。 |
| 53 | Scanner useDelimiter(String pattern) 此方法设置此扫描器的分隔模式，以从指定字符串构造的模式。 |
| 54 | Scanner useLocale(Locale locale) 此方法设置此扫描器的语言环境，以指定的语言环境。 |
| 55 | Scanner useRadix(int radix) 此方法设置此扫描器的默认基数为指定基数。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.ServiceLoader类 - Java.util包

**java.util.ServiceLoader** 类是一个简单的服务提供者加载设施。以下是关于ServiceLoader要点：

- 服务装载机总是在调用者的安全上下文中执行。受信任的系统代码通常应该调用这个类的方法，而他们返回，优越的安全上下文中的迭代器的方法。
- 这个类实例多个并发线程使用是不安全的。

## 类声明

以下是java.util.ServiceLoader类的声明：

```
public final class ServiceLoader<S>
   extends Object
      implements Iterable<S>
```

## 参数

以下是java.util.ServiceLoader类参数：

- S -- 这是服务于这个加载器加载的类型。

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Iterator<S> iterator() 此方法加载这个加载器的服务提供者。 |
| 2 | public static <S> ServiceLoader<S> load(Class<S> service) 此方法创建一个新的服务加载器给定服务类型，使用当前线程的上下文类加载器。 |
| 3 | public static <S> ServiceLoader<S> load(Class<S> service, ClassLoader loader) 此方法创建一个新的服务加载器给定服务类型和类加载器。 |
| 4 | public static <S> ServiceLoader<S> loadInstalled(Class<S> service) 此方法创建一个新的服务加载器给定服务类型，使用扩展类加载器。 |
| 5 | void reload() 此方法清除该加载器的服务者缓存，所有的提供者将被重新加载。 |
| 6 | String toString() 此方法返回描述此服务的字符串。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.SimpleTimeZone类 - Java.util包

**java.util.SimpleTimeZone** 类是时区的具体子类，它表示与公历使用的时区。以下是有关SimpleTimeZone的 要点：

- 这个类持有GMT的偏移，称为原始偏移。
- 这个类还拥有开始和结束的夏令时安排的规则。

## 类声明

以下是java.util.SimpleTimeZone类的声明：

```
public class SimpleTimeZone
   extends TimeZone
```

## 字段域

以下是java.util.SimpleTimeZone类中的字段：

- static int STANDARD_TIME -- 这是不变的开始或结束时间的规定为标准时间的模式。
- static int UTC_TIME -- 这是不变的开始或结束时间指定为UTC的模式。
- static int WALL_TIME -- 这是不变的开始或结束时间指定为挂钟时间的模式。

它还包括从类 TimeZone 继承的字段。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **SimpleTimeZone(int rawOffset, String ID)** 此构造一个SimpleTimeZone与给定基准时区GMT和时区的ID与保存白昼没有时间表偏移。 |
| 2 | **SimpleTimeZone(int rawOffset, String ID, int startMonth, int startDay, int startDayOfWeek, int startTime, int endMonth, int endDay, int endDayOfWeek, int endTime)** 此构造一个SimpleTimeZone与给定基准时区GMT偏移量，时区ID，以及启动和结束夏令时规则。 |
| 3 | **SimpleTimeZone(int rawOffset, String ID, int startMonth, int startDay, int startDayOfWeek, int startTime, int endMonth, int endDay, int endDayOfWeek, int endTime, int dstSavings)** 此构造一个SimpleTimeZone与给定基准时区GMT偏移量，时区ID，以及启动和结束夏令时规则。 |
| 4 | **SimpleTimeZone(int rawOffset, String ID, int startMonth, int startDay, int startDayOfWeek, int startTime, int startTimeMode, int endMonth, int endDay, int endDayOfWeek, int endTime, int endTimeMode, int dstSavings)** 此构造一个SimpleTimeZone与给定基准时区GMT偏移量，时区ID，以及启动和结束夏令时规则。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Object clone() 此方法返回当前的SimpleTimeZone实例的一个副本。 |
| 2 | boolean equals(Object obj) 此方法比较两个的SimpleTimeZone对象是否相等。 |
| 3 | int getDSTSavings() 此方法返回以毫秒为单位的时钟是在夏令时提前的时间量。 |
| 4 | int getOffset(int era, int year, int month, int day, int dayOfWeek, int millis) 此方法返回本地时间与UTC之间以毫秒为单位的差异，同时考虑到原始偏移量和夏令时的效果，对于指定的日期和时间。 |
| 5 | int getOffset(long date) 此方法返回当前时区从UTC偏移量在给定的时间。 |
| 6 | int getRawOffset() 此方法获取GMT这个时区偏移量。 |
| 7 | int hashCode() 此方法生成SimpleDateFormat对象的哈希代码。 |
| 8 | boolean hasSameRules(TimeZone other) 如果此区域具有相同的规则和偏移量为另一个区域，此方法返回true。 |
|  | boolean inDaylightTime(Date date) 此方法查询，如果给定的日期是在 |

| | |
|---|---|
| | 夏令时。 |
| 10 | void setDSTSavings(int millisSavedDuringDST) 此方法以毫秒为单位的时钟是在夏令时提前设定的时间量。 |
| 11 | void setEndRule(int endMonth, int endDay, int endTime) 此方法设置夏令时结束规则设置为某个月份的固定日期。 |
| 12 | void setEndRule(int endMonth, int endDay, int endDayOfWeek, int endTime) 此方法设置夏令时结束规则。 |
| 13 | void setEndRule(int endMonth, int endDay, int endDayOfWeek, int endTime, boolean after) 此方法设置夏令时结束规则到平日在一个月内给定日期之前或之后，如第一个星期一或以后的第8位。 |
| 14 | void setRawOffset(int offsetMillis) 此方法设置的基准时区偏移为GMT。 |
| 15 | void setStartRule(int startMonth, int startDay, int startTime) 此方法设置夏令时开始规则设置为某个月份的固定日期。 |
| 16 | void setStartRule(int startMonth, int startDay, int startDayOfWeek, int startTime) 此方法设置夏令时开始规则。 |
| 17 | void setStartRule(int startMonth, int startDay, int startDayOfWeek, int startTime, boolean after) 此方法之前或之后指定的日期在一个月之内，例如，在第一个星期一或以后的第8位。设置夏令时开始规则工作日 |
| 18 | void setStartYear(int year) 此方法设置夏令时的开始年份。 |
| 19 | String toString() 此方法返回当前时区的字符串表示形式。 |
| 20 | boolean useDaylightTime() 此方法查询，如果此时区使用夏令时。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.TimeZone
- java.util.Object

# java.util.Stack类 - Java.util包

**java.util.Stack** 类代表对象的后进先出(LIFO)堆栈。

- 当创建一个堆栈，它不包含任何项。
- 在这个类中，插入的最后一个元素会是第一个被访问。

## 类声明

以下是java.util.Stack类的声明：

```
public class Stack<E>
    extends Vector<E>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Stack()** 此构造函数创建一个空栈。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean empty() 此方法测试堆栈是否为空。 |
| 2 | E peek() 此方法会查看该对象在该堆栈的顶部而不从栈中删除。 |
| 3 | E pop() 此方法会删除在该堆栈的顶部的对象，并返回该对象作为该函数的值。 |
| 4 | E push(E item) 此方法推的项目到这个堆栈的顶部。 |
| 5 | int search(Object o) 此方法返回从1开始的位置，一个对象在栈中。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Vector

- java.util.AbstractList
- java.util.Object
- java.util.List

# java.util.StringTokenizer类 - Java.util包

**java.util.StringTokenizer** 类允许应用程序将字符串分解令牌。

- 由于类保留兼容性的原因，不鼓励使用新的代码遗留下来的类。
- 其方法不识别，数字和引用字符串中区分开来。
- 这个类的方法甚至不承认和跳过注释。

## 类声明

以下是java.util.StringTokenizer类的声明：

```
public class StringTokenizer
   extends Object
      implements Enumeration<Object>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **StringTokenizer(String str)** 此构造函数字符串标记生成器为指定的字符串。 |
| 2 | **StringTokenizer(String str, String delim)** 这个构造函数构造字符串标记为指定的字符串。 |
| 3 | **StringTokenizer(String str, String delim, boolean returnDelims)** 这个构造函数构造一个字符串标记为指定的字符串。 |

## 类方法

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | int countTokens() 此方法计算的时候，这个标记生成器的nextToken方法可以前它生成一个异常被调用的次数。 |
| 2 | boolean hasMoreElements() 此方法返回hasMoreTokens方法的相同的值。 |
| 3 | boolean hasMoreTokens() 如果此方法测试是否有从此标记生成的字符串提供更多的标记。 |
| 4 | Object nextElement() 此方法返回nextToken方法相同的值，不同之处在于它的声明的返回值是Object而不是String类型。 |
| 5 | String nextToken() 此方法从这个字符串标记生成器返回下一个标记。 |
| 6 | String nextToken(String delim) 此方法返回这个字符串标记生成的字符串中的下一个标记。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.Timer类 - Java.util包

java.util.Timer 类提供了工具，线程调度任务在后台线程中将来执行。

- 这个类是线程安全的，即多个线程可以无需进行外部同步共享单个Timer对象。
- 此类安排任务执行一次，或者定期重复执行。
- 所有构造函数启动一个计时器线程。

## 类声明

以下是java.util.Timer类的声明：

```
public class Timer
    extends Object
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **Timer()** 此构造函数创建一个新的计时器。 |
| 2 | **Timer(boolean isDaemon)** 此构造函数创建一个新计时器，其相关的线程可以被指定作为守护程序运行。 |
| 3 | **Timer(String name)** 此构造函数创建一个新计时器，其相关的线程具有指定的名称。 |
| 4 | **Timer(String name, boolean isDaemon)** 此构造函数创建一个新计时器，其相关的线程具有指定的名称，并且可以指定作为守护程序运行。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void cancel() 此方法终止此计时器，丢弃所有当前已安排的任务。 |
| 2 | int purge() 此方法从这个计时器的任务队列中移除所有已取消的任务。 |
| 3 | void schedule(TimerTask task, Date time) 此方法的时间表执行指定的任务在指定的时间。 |
| 4 | void schedule(TimerTask task, Date firstTime, long period) 此方法的时间表进行重复的固定延迟执行，开始指定的任务在指定的时间。 |
| 5 | void schedule(TimerTask task, long delay) 此方法的时间表指定的任务在指定的延迟后执行。 |
| 6 | void schedule(TimerTask task, long delay, long period) 此方法的时间表进行重复的固定延迟执行，在指定的延迟后开始指定的任务。 |
| 7 | void scheduleAtFixedRate(TimerTask task, Date firstTime, long period) 此方法的时间表进行重复的固定速率执行，开始在指定的时间指定的任务。 |
| 8 | void scheduleAtFixedRate(TimerTask task, long delay, long period) 此方法时间表进行重复的固定速率执行，在指定的延迟后开始指定的任务。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.TimerTask类 - Java.util包

**java.util.TimerTask** class represents a task that can be scheduled for one-time or repeated execution by a Timer.

## 类声明

Following is the declaration for **java.util.TimerTask** class:

```
public abstract class TimerTask
   extends Object
      implements Runnable
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **protected TimerTask()** This constructor creates a new timer task. |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean cancel() 此方法将取消此计时器任务。 |
| 2 | abstract void run() 此方法表示要由该定时任务执行的动作。 |
| 3 | long scheduledExecutionTime() 此方法返回当前任务最近实际执行的安排执行时间。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.TimeZone类 - Java.util包

**java.util.TimeZone**类表示时区偏移量，也可以计算夏令时。以下是关于时区的要点：

- 它考虑到了不同的时区。
- 通过此类别下使用的方法在任何一个国家运行的程序，获取基于特定国家的时区的时区对象。

## 类声明

以下是java.util.TimeZone类的声明：

```
public abstract class TimeZone
   extends Object
      implements Serializable, Cloneable
```

## 字段域

以下是java.util.TimeZone类中的字段：

- static int LONG -- 这是风格说明符getDisplayName()表示长的名字，如"太平洋标准时间"。
- static int SHORT -- 这是风格说明符getDisplayName()，表示一个简短的名称，如"太平洋标准时间"。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **TimeZone()** 这个构造函数是调用由子类构造了一个构造函数 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Object clone() 此方法创建这个TimeZone副本 |
| 2 | static String[] getAvailableIDs() 此方法获取所有支持的可用的ID。 |
| | |

| | |
|---|---|
| 3 | static String[] getAvailableIDs(int rawOffset) 根据给定的时区以毫秒为单位抵消此方法获取可用的ID。 |
| 4 | static TimeZone getDefault() 此方法获取的默认为这台主机的时区。 |
| 5 | String getDisplayName() 此方法返回适合于展示该时区的名称，在默认区域的用户。 |
| 6 | String getDisplayName(boolean daylight, int style) 此方法返回适合于展示该时区的名称，在默认区域的用户。 |
| 7 | String getDisplayName(boolean daylight, int style, Locale locale) 此方法返回适合于展示该时区的名称，在指定的区域的用户。 |
| 8 | String getDisplayName(Locale locale) 此方法返回适合于展示该时区的名称，在指定的区域的用户。 |
| 9 | int getDSTSavings() 此方法返回的时间为要添加到本地标准时间以获取本地挂钟时间。 |
| 10 | String getID() 此方法获取该时区的ID |
| 11 | abstract int getOffset(int era, int year, int month, int day, int dayOfWeek, int milliseconds) 此方法获取的时区偏移量，为当前日期，修改夏令时情况下。 |
| 12 | int getOffset(long date) 此方法返回从UTC偏移量在指定的日期的当前时区。 |
| 13 | abstract int getRawOffset() 此方法返回的时间以毫秒为单位添加到UTC以获取标准时间在这个时区。 |
| 14 | static TimeZone getTimeZone(String ID) 此方法获取的时区为给定的ID。 |
| 15 | boolean hasSameRules(TimeZone other) 如果此区域具有相同的规则和偏移量为另一个区域，此方法返回true。 |
| 16 | abstract boolean inDaylightTime(Date date) 此方法查询，给定的日期是否在夏令时在该时区。 |
| 17 | static void setDefault(TimeZone zone) 此方法由getDefault方法设置时区，方法返回设置的时区。 |
| 18 | void setID(String ID) 此方法设置的时区ID |
| 19 | abstract void setRawOffset(int offsetMillis) 此方法为GMT设置的基准时区偏移。 |
| 20 | abstract boolean useDaylightTime() 此方法查询，此时区是否使用夏令时。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.TreeMap类 - Java.util包

java.util.TreeMap 类是Red-Black树实现基于Map接口。以下是关于TreeMap中重要的几点：

- TreeMap类保证该映射将是升序键顺序。
- 该映射是按照自然排序方法的关键类，或者根据创建映射时提供的比较器，这将取决于其构造函数中使用排序。

## 类声明

以下是java.util.TreeMap类的声明：

```
public class TreeMap<K,V>
   extends AbstractMap<K,V>
      implements NavigableMap<K,V>, Cloneable, Serializable
```

## 参数

以下是java.util.TreeMap类的参数：

- K -- 这是此映射维护的键的类型。
- V -- 这是映射值的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **TreeMap()** 这个构造函数构造一个新的，空的树映射，使用键的自然顺序。 |
| 2 | **TreeMap(Comparator<? super K> comparator)** 这个构造函数构造一个新的，空的树映射，根据给定的比较器进行排序。 |
| 3 | **TreeMap(Map<? extends K,? extends V> m)** 这个构造函数构造一个新的树映射具有相同映射关系为给定的映射，根据其键的自然顺序进行排序。 |
| 4 | **TreeMap(SortedMap<K,? extends V> m)** 这个构造函数构造一个新的树映射具有相同映射关系，并使用相同的顺序在指定的有序映射。 |

# 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | Map.Entry<K,V> ceilingEntry(K key) 此方法返回的最小键大于等于给定键，则返回null关联的键 - 值映射关系，如果不存在这样的键。 |
| 2 | K ceilingKey(K key) 此方法返回大于最小键大于或等于给定键，或null，如果不存在这样的键。 |
| 3 | void clear() 此方法移除此映射中的所有项。 |
| 4 | Object clone() 此方法返回这个TreeMap实例的浅表副本。 |
| 5 | Comparator<? super K> comparator() 此方法返回用于排序的关键在此映射的比较器，则返回null，如果此映射使用键的自然顺序。 |
| 6 | boolean containsKey(Object key) 如果此映射包含指定键的映射，此方法返回true。 |
| 7 | boolean containsValue(Object value) 如果此映射将一个或多个键映射到指定值，该方法返回true。 |
| 8 | NavigableSet<K> descendingKeySet() 此方法返回此映射中包含的键的逆序NavigableSet视图。 |
| 9 | NavigableMap<K,V> descendingMap() 此方法返回此映射中包含的映射关系的逆序视图。 |
| 10 | Set<Map.Entry<K,V>> entrySet() 此方法返回此映射中包含的映射关系的Set视图。 |
| 11 | Map.Entry<K,V> firstEntry() 此方法返回的最小键在这个映射中的键- 值映射关系，如果映射为空，返回null。 |
| 12 | K firstKey() 该方法返回此映射第一个（最低）键。 |
| 13 | Map.Entry<K,V> floorEntry(K key) 此方法返回的最大键小于等于给定键关联的键- 值映射关系，则返回null，如果不存在这样的键 |
| 14 | K floorKey(K key) 此方法返回比最大键小于或等于给定键，或null，如果不存在这样的键 |
| 15 | V get(Object key) 此方法返回指定键所映射的值，则返回null如果此映射不包含该键。 |
| 16 | SortedMap<K,V> headMap(K toKey) 此方法返回此映射的键严格小于toKey的部分视图。 |
| 17 | NavigableMap<K,V> headMap(K toKey, boolean inclusive) 此方法返回此映射，其键小于toKey的部分视图（或等于，如果inclusive为true）。 |
| 18 | Map.Entry<K,V> higherEntry(K key) 此方法返回的返回具有最小键严格大于给定键的关联的键 - 值映射关系，则返回null，如果不存在这样的 |

|  | 键 |
|---|---|
| 19 | K higherKey(K key) 此方法返回最小键严格大于给定键，则返回null，如果不存在这样的键 |
| 20 | Set<K> keySet() 此方法返回此映射中包含的键的Set视图。 |
| 21 | Map.Entry<K,V> lastEntry() 此方法返回此映射中的最大键的关联的键-值映射关系，如果映射为空，返回null。 |
| 22 | K lastKey() 该方法返回此映射最后一个（最高）键。 |
| 23 | Map.Entry<K,V> lowerEntry(K key) 此方法返回的最大键严格小于给定键的关联的键- 值映射关系，则返回null，如果不存在这样的键。 |
| 24 | K lowerKey(K key) 此方法返回的最大键严格小于给定键，如果不存在这样的键,则返回null。 |
| 25 | NavigableSet<K> navigableKeySet() 此方法返回此映射中包含的键的NavigableSet视图。 |
| 26 | Map.Entry<K,V> pollFirstEntry() 此方法删除，并返回具有最小键在此映射中关联的键- 值映射关系。如果映射为空，返回null。 |
| 27 | Map.Entry<K,V> pollLastEntry() 该方法移除并返回与此映射中的最大键的关联的键 - 值映射关系，如果映射为空，返回null。 |
| 28 | V put(K key, V value) 该方法将指定值与此映射中的指定键。 |
| 29 | void putAll(Map<? extends K,? extends V> map) 该方法会将所有从指定映射到此映射中。 |
| 30 | V remove(Object key) 该方法移除了映射关系，如果TreeMap中存在该键。 |
| 31 | int size() 此方法返回此映射中的键 - 值映射关系的数量。 |
| 32 | NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive) 此方法返回此映射的键的部分视图，范围从fromKey到toKey |
| 33 | SortedMap<K,V> subMap(K fromKey, K toKey) 此方法返回此映射的键值从fromKey（包括）到toKey（不包括）的部分视图。 |
| 34 | SortedMap<K,V> tailMap(K fromKey) 此方法返回此映射，其键大于或等于fromKey的部分视图。 |
| 35 | NavigableMap<K,V> tailMap(K fromKey, boolean inclusive) 此方法返回此映射，其键大于fromKey的部分视图（或等于，如果inclusive为true）。 |
| 36 | Collection<V> values() 此方法返回此映射中包含的值的Collection视图。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.AbstractMap
- java.util.Object
- java.util.Map

# java.util.TreeSet类 - Java.util包

java.util.TreeSet 类实现Set接口。以下是关于TreeSet的要点：

- TreeSet类保证该映射将在升序键顺序，由TreeMap支持。
- 该映射是按照自然排序方法该键类，或在集创建时提供的比较器，这将取决于其构造函数中使用排序。
- 顺序必须是总为了使树到功能属性。

## 类声明

以下是java.util.TreeSet类声明：

```
public class TreeSet<E>
   extends AbstractSet<E>
      implements NavigableSet<E>, Cloneable, Serializable
```

## 参数

以下是java.util.TreeSet中类参数：

- E -- 这是由该组保持元件的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
| --- | --- |
| 1 | **TreeSet()** 此构造一个新的空树集，根据其元素的自然顺序进行排序。 |
| 2 | **TreeSet(Collection<? extends E> c)** 这个构造函数构造一个包含指定集合，根据其元素的自然顺序进行排序的元素的新树集。 |
| 3 | **TreeSet(Comparator<? super E> comparator)** 此构造一个新的空树集，根据指定的比较器进行排序。 |
| 4 | **TreeSet(SortedSet<E> s)** 此构造包含相同的元素，并使用相同的顺序与指定有序集的新树集。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | boolean add(E e) 此方法将指定的元素来添加此set，如果它是不存在。 |
| 2 | boolean addAll(Collection<? extends E> c) 此方法将所有指定collection到此set元素。 |
| 3 | E ceiling(E e) 此方法返回的最小元素此设定为大于或等于给定的元素，或null，如果不存在这样的元素。 |
| 4 | void clear() 此方法移除此集合中的元素。 |
| 5 | Object clone() 此方法返回这个TreeSet的实例的浅表副本。 |
| 6 | Comparator<? super E> comparator() 此方法返回用于排序在此set中，或返回null，如果此set使用其元素的自然顺序比较。 |
| 7 | boolean contains(Object o) 如果此set包含指定的元素此方法返回true。 |
| 8 | Iterator<E> descendingIterator() 此方法返回一个迭代器在此set降序排列的元素。 |
| 9 | NavigableSet<E> descendingSet() 此方法返回包含在这个集合中元素的逆序视图。 |
| 10 | E first() 此方法在此set目前正在返回第一个（最低）元素。 |
| 11 | E floor(E e) 此方法返回在此的最大元素设置为小于或等于给定的元素，或null，如果不存在这样的元素。 |
| 12 | SortedSet<E> headSet(E toElement) 这个方法返回这个集合，其元素严格小于toElement的部分视图。 |
| 13 | NavigableSet<E> headSet(E toElement, boolean inclusive) 此方法返回这个集合的元素是小于toElement的部分视图（或等于，如果inclusive为true）。 |
| 14 | E higher(E e) 此方法返回的最小元素此设定严格大于给定的元素，或null，如果不存在这样的元素。 |
| 15 | boolean isEmpty() 如果此set不包含元素，此方法返回true。 |
| 16 | Iterator<E> iterator() 此方法返回一个迭代器在此set升序排列元素。 |
| 17 | E last() 这种方法在这组目前正在返回最后一个（最高）元素。 |
| 18 | E lower(E e) 此方法返回此set中严格小于给定的最大元素，返回null，如果没有这样的元素。 |
| 19 | E pollFirst() 此方法检索并移除第一个（最低）元素;如果此set为空，则返回null。 |
| 20 | E pollLast() 此方法检索并移除最后一个（最高）元素;如果此set为空，则返回null。 |
| 21 | boolean remove(Object o) 该方法将删除该组指定元素（如果存在）。 |

| | |
|---|---|
| 22 | int size() 这个方法返回这个集合（其容量）的元素个数。 |
| 23 | NavigableSet<E> subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) 这个方法返回这个集合，其元素范围从fromElement到toElement的部分视图。 |
| 24 | SortedSet<E> subSet(E fromElement, E toElement) 这个方法返回这个集合，其元素范围从fromElement（包括）到toElement（不包括）的部分视图。 |
| 25 | SortedSet<E> tailSet(E fromElement) 这个方法返回这个集合，其元素大于或等于fromElement的部分视图。 |
| 26 | NavigableSet<E> tailSet(E fromElement, boolean inclusive) 这个方法返回这个集合，其元素大于fromElement的部分视图（或等于，如果inclusive为true）。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractSet
- java.util.AbstractCollection
- java.util.Object
- java.util.Set

# java.util.UUID类 - Java.util包

**java.util.UUID** 类表示一个不变的通用唯一标识符(UUID)。以下是有关UUID的要点：

- 一个UUID表示一个128位的值。
- 它是用于创建随机文件名，在Web应用程序的会话ID，事务ID等。
- 还有UUID四种不同的基本类型：基于时间，DCE安全性，基于名称，并随机生成的UUID。

## 类声明

以下是java.util.UUID类的声明：

```
public final class UUID
   extends Object
      implements Serializable, Comparable<UUID>
```

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **UUID(long mostSigBits, long leastSigBits)** 此构造使用指定的数据一个新的UUID。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | int clockSequence() 此方法显示了与此UUID相关联的时钟序列值。 |
| 2 | int compareTo(UUID val) 此方法比较此UUID与指定的UUID。 |
| 3 | boolean equals(Object obj) 此方法比较此对象与指定对象。 |
| 4 | static UUID fromString(String name) 此方法创建的字符串标准表示一个UUID。 |
| 5 | long getLeastSignificantBits() 此方法返回此UUID的128位至少显著64位值。 |
| 6 | long getMostSignificantBits() 此方法返回此UUID的128位最显著64位值。 |
| 7 | int hashCode() This method returns a hash code for this UUID. |
| 8 | static UUID nameUUIDFromBytes(byte[] name) 静态工厂来获取一个类型3（基于名称的），根据指定的字节数组的UUID。 |
| 9 | long node() 此方法返回与此UUID相关联的节点值。 |
| 10 | static UUID randomUUID() 静态工厂来获取一个类型4（伪随机生成）的UUID。 |
| 11 | long timestamp() 此方法返回与此UUID相关联的时间戳值。 |
| 12 | String toString() 此方法返回表示此UUID的String对象。 |
| 13 | int variant() 此方法返回与此UUID相关联的变型数量。 |
| 14 | int version() 此方法消除了与此UUID相关联的版本号。 |

# 方法继承

这个类从以下类继承的方法：

- java.util.Object

# java.util.WeakHashMap类 - Java.util包

**java.util.WeakHashMap** 类是基于哈希表Map类实现了弱密钥。 WeakHashMap中的条目将自动被垃圾收集器，当其键不再使用会被删除。以下是关于WeakHashMap的要点：

- 无论是null值和null键都支持。
- 像大多数集合类，这个类也没有同步。
- 这个类主要用于与主要对象的equals方法测试，使用==操作符对象标识的使用。
- WeakHashMap每个键对象间接地存储为一个弱引用的引用。
- 这个类是Java集合框架成员。

## 类声明

以下是java.util.WeakHashMap类的声明：

```
public class WeakHashMap<K,V>
   extends AbstractMap<K,V>
      implements Map<K,V>
```

在这里，<K>键是由该映射和<V>映射值的类型保持的类型。

## 类构造函数

| S.N. | 构造函数 & 描述 |
|---|---|
| 1 | **WeakHashMap()** 此构造函数用于创建具有默认初始容量（16）和加载因子（0.75）在空的WeakHashMap中。 |
| 2 | **WeakHashMap(int initialCapacity)** 此构造函数用于创建具有给定的初始容量和默认加载因子（0.75）在空的WeakHashMap中。 |
| 3 | **WeakHashMap(int initialCapacity, float loadFactor)** 此构造函数用于创建具有给定的初始容量和给定的负载因子在空的WeakHashMap中。 |
| 4 | **WeakHashMap(Map<? extends K,? extends V> m)** 此构造函数用于创建一个新WeakHashMap中具有相同的映射关系与指定映射。 |

## 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | void clear() 此方法移除此映射中。 |
| 2 | boolean containsKey(Object key) 如果此映射包含指定键的映射此方法返回true。 |
| 3 | boolean containsValue(Object value) 如果此映射将一个或多个键映射到指定值， 该方法返回true。 |
| 4 | Set<Map.Entry>K,V>> entrySet() 此方法返回此映射中包含的映射关系的Set视图。 |
| 5 | v get(Object key) 此方法返回指定键所映射的值， 则返回null如果此映射不包含该键。 |
| 6 | boolean isEmpty() 如果此映射不包含键 - 值映射关系， 此方法返回true。 |
| 7 | Set<K> keySet() 此方法返回此映射中包含的键的Set视图。 |
| 8 | v put(K key, V value) 此方法将指定值与此映射中的指定的键。 |
| 9 | void putAll(Map<? extends K,? extends V> m) 此方法会将所有从指定映射到此映射中。 |
| 10 | v remove(Object key) 此方法删除映射关系， 这种弱哈希映射中的键（如果存在）。 |
| 11 | int size() 此方法返回此映射中的键 - 值映射关系数量。 |
| 12 | Collection<V> values() 此方法返回此映射中包含的值的Collection视图。 |

## 方法继承

这个类从以下类继承的方法：

- java.util.AbstractMap
- java.lang.Object
- java.util.Map

# java.util.Interfaces接口 - Java.util包

java.util.Interfaces 包含集合框架，遗留的collection类，事件模型，日期和时间，国际化和各种实用工具类(字符串标记生成器，随机数生成器和位数组)。

## 接口概要

| S.N. | 接口与说明 |
| --- | --- |
| 1 | **Collection<E>** 这是在集层次结构的根接口。 |
| 2 | **Comparator<T>** 这是一个比较函数强行对某些对象collection进行整体排序。 |
| 3 | **Deque<E>** 这是支持元素插入和移除在两端一个线性的集合。 |
| 4 | **Enumeration<E>** 这是一个对象，它实现Enumeration接口生成一系列元素，一次一个。 |
| 5 | **EventListener** 这是一个标记接口的所有事件侦听器接口必须扩展。 |
| 6 | **Formattable** 这是Formattable接口必须由需要执行自定义使用格式化的's'转换说明符的任何类中实现。 |
| 7 | **Iterator<E>** 这是一个迭代器的集合。 |
| 8 | **List<E>** 这是一个有序集合(也称为一个序列)。 |
| 9 | **ListIterator<E>** 这是表迭代器，允许程序员来遍历列表中任一方向，迭代期间修改列表，并获得该列表中的迭代器的当前位置。 |
| 10 | **Map<K,V>** 这是一个键映射到值的对象。 |
| 11 | **Map.Entry<K,V>** 这是一个映射项(键 - 值对)。 |
| 12 | **NavigableMap<K,V>** 这是扩展了导航方法返回给定搜索目标最接近的匹配一个SortedMap。 |
| 13 | **NavigableSet<E>** 这是扩展了导航方法报告给定搜索目标最接近的匹配项的SortedSet。 |
| 14 | **Observer** 这是一个类可以实现Observer接口时，它希望被告知改变观察的对象。 |
| 15 | **Queue<E>** 这是设计处理之前持有的元素的集合。 |
| 16 | **RandomAccess()** 这是List实现所使用，以表明其支持快速(通常是固定时间)随机访问标记接口。 |
| 17 | **Set<E>** 这是不包含重复的元素的集合。 |
| 18 | **SortedMap<K,V>** 这是一个映射还提供了对键的总体排序。 |
| 19 | **SortedSet<E>** 这是一组还提供了关于其元素的总排序。 |

# java.util.Exceptions接口 - Java.util包

**java.util.Exceptions** 包含集合框架，遗留的collection类，事件模型，日期和时间，国际化和各种实用工具类(字符串标记生成器，随机数生成器和位数组)。

## 接口概要

| S.N. | 接口与说明 |
|---|---|
| 1 | **ConcurrentModificationException** 此异常可能由已检测到对象的并发修改时，这种修改是不允许的方法抛出。 |
| 2 | **DuplicateFormatFlagsException** 这是未经检查的异常时，在格式说明符中提供重复标志抛出。 |
| 3 | **EmptyStackException** 这是通过在Stack类的方法抛出，指示堆栈为空。 |
| 4 | **FormatFlagsConversionMismatchException** 这是未经检查的异常抛出时，转换和标志是不相容的。 |
| 5 | **FormatterClosedException** 这是未经检查的异常抛出时，格式化已被关闭。 |
| 6 | **IllegalFormatCodePointException** 这是未经检查的异常时，通过Character.isValidCodePoint（int）定义了一个无效的Unicode代码点的字符传递给格式化抛出。 |
| 7 | **IllegalFormatConversionException** 这是未经检查的异常抛出时，对应于格式说明符的参数是不兼容的类型。 |
| 8 | **IllegalFormatException** 这是未经检查的异常时抛出一个格式字符串包含非法语法或格式说明符是与给定参数不兼容。 |
| 9 | **IllegalFormatFlagsException** 这是未经检查的异常抛出一个非法组合标志时给出。 |
| 10 | **IllegalFormatPrecisionException** 这是未经检查的异常抛出时的精度是其他的负值比-1，转换不支持某个精度或者值在其他方面不受支持。 |
| 11 | **IllegalFormatWidthException** 这是未经检查的异常时抛出的幅面宽度为负值除-1以外或其他不受支持。 |
| 12 | **InputMismatchException** 这是由扫描器抛出，指示检索到的令牌不匹配的格局，为预期的类型，或者该标记超出范围的预期的类型。 |
| 13 | **InvalidPropertiesFormatException** 这被抛出，指示操作无法完成，因为输入不符合相应的XML文档类型的属性的集合，按照Properties规范。 |

| | |
|---|---|
| 14 | **MissingFormatArgumentException** 这是未经检查的异常抛出时，有一个格式说明符没有相应的参数，或者参数索引引用了不存在的说法。 |
| 15 | **MissingFormatWidthException** 这是未经检查的异常时，格式化宽度必须抛出。 |
| 16 | **MissingResourceException** 这标志着一个资源丢失。 |
| 17 | **NoSuchElementException** 这是通过枚举的nextElement方法抛出，表明有枚举没有更多的元素。 |
| 18 | **TooManyListenersException** 这是用来作为Java事件模型的一部分注释和实施的多播事件源的单播特例。 |
| 19 | **UnknownFormatConversionException** 这是未经检查的异常抛出一个未知的转换时给出。 |
| 20 | **UnknownFormatFlagsException** 这是未经检查的异常抛出一个未知的标志时给出。 |

# java.util.Formatter.BigDecimalLayoutForm接口 - Java.util包

**java.util.Formatter.BigDecimalLayoutForm** 枚举提供了可用的样式格式非常大的十进制数。

## Enum declaration

以下是java.util.Formatter.BigDecimalLayoutForm枚举的声明：

```
public static enum Formatter.BigDecimalLayoutForm
   extends Enum<Formatter.BigDecimalLayoutForm>
```

## Constants

以下是java.util.Formatter.BigDecimalLayoutForm枚举常量：

- DECIMAL_FLOAT -- 正常使用十进制/浮点数风格表示BigDecimals。
- SCIENTIFIC -- 用科学的风格表示BigDecimals。

## 枚举方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **static Formatter.BigDecimalLayoutForm valueOf(String name)** 此方法返回这个类型具有指定名称的枚举常量。 |
| 2 | **static Formatter.BigDecimalLayoutForm[] values()** 此方法返回一个包含该枚举类型的常量数组的顺序被声明。 |

# Java XML教程

XML（可扩展标记语言）是一种很流行的简单的基于文本的语言来用作应用程序之间的通信模式。它被认为是传输标准装置和存储数据。JAVA提供了极好的支持和丰富的库来解析，修改或查询XML文档。

## XML是什么?

XML是一种简单的基于文本的语言，它被设计为储存和运输以纯文本格式的数据。它代表着可扩展标记语言。以下是一些XML的显着特征。

- XML是一种标记语言。
- XML是一种标记语言就像HTML一样。
- XML标签不是像HTML那样预定义。
- 可以定义自己的标签，这就是为什么它被称为可扩展的语言。
- XML标签被设计成自描述性的。
- XML是W3C推荐用于数据存储和传输。

## 示例

```
<?xml version="1.0"?>
<Class>
   <Name>First</Name>
   <Sections>
      <Section>
         <Name>A</Name>
         <Students>
            <Student>Rohan</Student>
            <Student>Mohan</Student>
            <Student>Sohan</Student>
            <Student>Lalit</Student>
            <Student>Vinay</Student>
         </Students>
      </Section>
      <Section>
         <Name>B</Name>
         <Students>
            <Student>Robert</Student>
            <Student>Julie</Student>
            <Student>Kalie</Student>
            <Student>Michael</Student>
         </Students>
      </Section>
   </Sections>
</Class>
```

## 优势

以下是XML提供的优势：

- 技术无关 - 作为普通文本，XML是技术独立。它可以用于由任何技术进行数据的存储和传输的目的。
- 人类可读 - XML使用简单的文本格式。它是人类可读和可以理解的。
- 可扩展性 - 在XML，自定义标签可以创建和很容易使用。
- 允许验证 - 使用XSD，DTD和XML结构可以很容易地验证。

## 缺点

下面是使用XML的缺点：

- 冗余的语法 - 通常XML文件中包含大量的重复计算。
- 冗余 - 作为一个冗长的语言，XML文件大小增加了传输和存储成本。

# Java XML解析器 - Java XML教程

## 什么是XML解析？

解析XML是指将通过XML文档访问数据或修改数据的一个操作或方法。

## XML解析器是什么？

XML解析器提供方法来访问或修改XML文档中的数据。 Java提供了多种选择来解析XML文档。以下是各种类型解析器其通常用于解析XML文档。

- **Dom解析器** - 解析通过加载该文件的全部内容，并创建其完整分级树中存储的文件。
- **SAX解析器** - 解析基于事件触发器的文档。不完整(部分)的文件加载到存储器中。
- **JDOM解析器** - 解析以类似的方式，以DOM解析器但更简单的方法的文档。
- **StAX解析器** - 解析以类似的方式，以SAX解析器但在更高效的方式的文档。
- **XPath解析器** - 解析基于表达式XML并广泛选择使用XSLT。
- **DOM4J解析器** - Java库来解析XML，XPath和使用Java集合框架XSLT，为DOM，SAX和JAXP的支持。

JAXB和XSLT的API来处理XML解析在面向对象方法。我们将详细一个一个阐述解析器在下一章节。

# Java DOM解析器 - Java XML教程

文档对象模型是万维网联盟(W3C)的官方推荐。它定义了一个接口，使程序能够访问和更新样式，结构和XML文档的内容。支持DOM实现该接口的XML解析器。

## 何时使用？

在以下几种情况时，应该使用DOM解析器：

- 需要知道很多关于文档的结构
- 需要将文档的部分周围(例如，可能需要某些元素进行排序)
- 需要使用的文件中的信息超过一次

## 会得到什么？

当使用DOM 解析器解析一个XML文档，会得到一个树形结构，其中包含的所有文档的元素。 DOM提供了多种可用于检查文档的内容和结构的函数。

### 优势

DOM是用于处理文档结构的通用接口。它的一个设计目标是Java代码编写一个DOM兼容的解析器，运行在任何其他的DOM兼容的解析器不会有变化。

## DOM接口

DOM定义了几个Java接口。这里是最常见的接口：

- 节点 - DOM的基本数据类型。
- 元素 - 要处理的对象绝大多数是元素。
- **Attr** - 代表元素的属性。
- 文本 - 元素或Attr的实际内容。
- 文档 - 代表整个XML文档。文档对象是通常被称为DOM树。

## 常见的DOM方法

当正在使用DOM，有经常用到的几种方法：

- **Document.getDocumentElement()** - 返回文档的根元素。
- **Node.getFirstChild()** - 返回给定节点的第一个子节点。
- **Node.getLastChild()** - 返回给定节点的最后一个子节点。
- **Node.getNextSibling()** - 这些方法返回一个特定节点的下一个兄弟节点。
- **Node.getPreviousSibling()** - 这些方法返回一个特定节点的前一个兄弟节点。
- **Node.getAttribute(attrName)** - 对于给定的节点，则返回所请求的名字的属性。

# Java DOM解析器 - 解析XML文档 - Java XML教程

## 使用DOM的步骤

以下是在使用DOM解析器解析文档使用的步骤。

- 导入XML相关的软件包。
- 创建DocumentBuilder
- 从文件或流创建一个文档
- 提取根元素
- 检查属性
- 检查子元素

导入XML相关的软件包

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;
```

创建 **DocumentBuilder**

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
```

从文件或流创建一个文档

```
StringBuilder xmlStringBuilder = new StringBuilder();
xmlStringBuilder.append("<?xml version="1.0"?> <class> </class>");
ByteArrayInputStream input =  new ByteArrayInputStream(
   xmlStringBuilder.toString().getBytes("UTF-8"));
Document doc = builder.parse(input);
```

提取根元素

```
Element root = document.getDocumentElement();
```

检查属性

```
//returns specific attribute
getAttribute("attributeName");
//returns a Map (table) of names/values
getAttributes();
```

检查子元素

```
//returns a list of subelements of specified name
getElementsByTagName("subelementName");
//returns a list of all child nodes
getChildNodes();
```

# 演示示例

这是输入需要解析的 xml 文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

演示示例：

*DomParserDemo.java*

```
package com.yiibai.xml;
```

```
import java.io.File;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

public class DomParserDemo {
   public static void main(String[] args){

      try {
         File inputFile = new File("input.txt");
         DocumentBuilderFactory dbFactory
            = DocumentBuilderFactory.newInstance();
         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
         Document doc = dBuilder.parse(inputFile);
         doc.getDocumentElement().normalize();
         System.out.println("Root element :"
            + doc.getDocumentElement().getNodeName());
         NodeList nList = doc.getElementsByTagName("student");
         System.out.println("----------------------------");
         for (int temp = 0; temp < nList.getLength(); temp++) {
            Node nNode = nList.item(temp);
            System.out.println("\nCurrent Element :"
               + nNode.getNodeName());
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
               Element eElement = (Element) nNode;
               System.out.println("Student roll no : "
                  + eElement.getAttribute("rollno"));
               System.out.println("First Name : "
                  + eElement
                  .getElementsByTagName("firstname")
                  .item(0)
                  .getTextContent());
               System.out.println("Last Name : "
               + eElement
                  .getElementsByTagName("lastname")
                  .item(0)
                  .getTextContent());
               System.out.println("Nick Name : "
               + eElement
                  .getElementsByTagName("nickname")
                  .item(0)
                  .getTextContent());
               System.out.println("Marks : "
               + eElement
                  .getElementsByTagName("marks")
                  .item(0)
                  .getTextContent());
            }
         }
      } catch (Exception e) {
```

```
            e.printStackTrace();
         }
      }
   }
```

这将产生以下结果：

```
Root element :class
----------------------------

Current Element :student
Student roll no : 393
First Name : dinkar
Last Name : kad
Nick Name : dinkar
Marks : 85

Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
Nick Name : vinni
Marks : 95

Current Element :student
Student roll no : 593
First Name : jasvir
Last Name : singn
Nick Name : jazz
Marks : 90
```

# Java DOM解析器 - 查询XML文档 - Java XML教程

## 演示示例

这是需要我们查询的输入XML文件：

```
<?xml version="1.0"?>
<cars>
   <supercars company="Ferrari">
      <carname type="formula one">Ferarri 101</carname>
      <carname type="sports car">Ferarri 201</carname>
      <carname type="sports car">Ferarri 301</carname>
   </supercars>
   <supercars company="Lamborgini">
      <carname>Lamborgini 001</carname>
      <carname>Lamborgini 002</carname>
      <carname>Lamborgini 003</carname>
   </supercars>
   <luxurycars company="Benteley">
      <carname>Benteley 1</carname>
      <carname>Benteley 2</carname>
      <carname>Benteley 3</carname>
   </luxurycars>
</cars>
```

演示示例：

*QueryXmlFileDemo.java*

```
package com.yiibai.xml;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;

public class QueryXmlFileDemo {

   public static void main(String argv[]) {

      try {
         File inputFile = new File("input.txt");
         DocumentBuilderFactory dbFactory =
            DocumentBuilderFactory.newInstance();
```

```
         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder()
         Document doc = dBuilder.parse(inputFile);
         doc.getDocumentElement().normalize();
         System.out.print("Root element: ");
         System.out.println(doc.getDocumentElement().getNodeName()
         NodeList nList = doc.getElementsByTagName("supercars");
         System.out.println("----------------------------");
         for (int temp = 0; temp < nList.getLength(); temp++) {
            Node nNode = nList.item(temp);
            System.out.println("\nCurrent Element :");
            System.out.print(nNode.getNodeName());
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
               Element eElement = (Element) nNode;
               System.out.print("company : ");
               System.out.println(eElement.getAttribute("company")
               NodeList carNameList =
                  eElement.getElementsByTagName("carname");
               for (int count = 0;
                  count < carNameList.getLength(); count++) {
                  Node node1 = carNameList.item(count);
                  if (node1.getNodeType() ==
                     node1.ELEMENT_NODE) {
                     Element car = (Element) node1;
                     System.out.print("car name : ");
                     System.out.println(car.getTextContent());
                     System.out.print("car type : ");
                     System.out.println(car.getAttribute("type"));
                  }
               }
            }
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

这将产生以下结果：

```
Root element :cars
----------------------------

Current Element :supercars
company : Ferrari
car name : Ferarri 101
car type : formula one
car name : Ferarri 201
car type : sports car
car name : Ferarri 301
car type : sports car

Current Element :supercars
company : Lamborgini
car name : Lamborgini 001
car type :
car name : Lamborgini 002
car type :
car name : Lamborgini 003
car type :
```

# Java DOM解析器 - 修改XML文档 - Java XML教程

## 演示示例

这是我们需要修改的输入XML文件：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars>
   <supercars company="Ferrari">
      <carname type="formula one">Ferrari 101</carname>
      <carname type="sports">Ferrari 202</carname>
   </supercars>
   <luxurycars company="Benteley">
      <carname>Benteley 1</carname>
      <carname>Benteley 2</carname>
      <carname>Benteley 3</carname>
   </luxurycars>
</cars>
```

演示示例：

*ModifyXmlFileDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class ModifyXmlFileDemo {

public static void main(String argv[]) {

   try {
      File inputFile = new File("input.xml");
      DocumentBuilderFactory docFactory =
      DocumentBuilderFactory.newInstance();
      DocumentBuilder docBuilder =
```

```
         docFactory.newDocumentBuilder();
         Document doc = docBuilder.parse(inputFile);
         Node cars = doc.getFirstChild();
         Node supercar = doc.getElementsByTagName("supercars").item(0)
         // update supercar attribute
         NamedNodeMap attr = supercar.getAttributes();
         Node nodeAttr = attr.getNamedItem("company");
         nodeAttr.setTextContent("Lamborigini");

         // loop the supercar child node
         NodeList list = supercar.getChildNodes();
         for (int temp = 0; temp < list.getLength(); temp++) {
            Node node = list.item(temp);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
               Element eElement = (Element) node;
               if ("carname".equals(eElement.getNodeName())){
                  if("Ferrari 101".equals(eElement.getTextContent()))
                     eElement.setTextContent("Lamborigini 001");
                  }
               if("Ferrari 202".equals(eElement.getTextContent()))
                  eElement.setTextContent("Lamborigini 002");
               }
            }
         }
         NodeList childNodes = cars.getChildNodes();
         for(int count = 0; count < childNodes.getLength(); count++){
            Node node = childNodes.item(count);
            if("luxurycars".equals(node.getNodeName()))
               cars.removeChild(node);
            }
            // write the content on console
            TransformerFactory transformerFactory =
            TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransforme
            DOMSource source = new DOMSource(doc);
            System.out.println("-----------Modified File-----------")
            StreamResult consoleResult = new StreamResult(System.out)
            transformer.transform(source, consoleResult);
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
```

这将产生以下结果：

```
-----------Modified File-----------
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars>
<supercars company="Lamborigini">
<carname type="formula one">Lamborigini 001</carname>
<carname type="sports">Lamborigini 002</carname>
</supercars></cars>
```

# Java SAX解析器 - Java XML教程

SAX(针对XML的简单API)是基于事件为XML文档的解析器。不像DOM解析器，SAX解析器创建没有解析树。 SAX是一个流接口用于XML的，这意味着使用SAX应用接收事件通知有关XML文档被处理的元素，属性，在按顺序每次开始在文档的顶部，并与所述闭合结束根元素。

- 读取XML文件从上到下，构成一个结构完整的XML文档的标记
- 令牌以相同的顺序进行处理，它们出现在文档中
- 报告应用程序，因为它们所出现解析器遇到标记的特性
- 应用程序提供了必须的解析器注册的"事件"处理程序
- 作为标记标识，在处理程序回调方法相关信息调用

## 什么时候使用？

应该使用SAX解析器的时候：

- 可以在XML文档从上往下处理以线性方式
- 该文件并不深层次嵌套
- 处理一个非常大的XML文档，DOM树会占用太多的内存。典型DOM的实现使用10字节的存储器以表示XML的一个字节
- 解决的问题涉及的XML文档的一部分
- 数据是可用的，只要它是由解析器看出，这样的SAX可以很好地用于到达流的XML文档

## SAX的缺点

- 它是在一个只进入处理随机访问方式XML文档
- 如果需要跟踪的数据分析器已经看到或更改项目的顺序，必须自已编写代码和数据存储

## ContentHandler接口

此接口指定SAX解析器用来通知XML文档，已经看到部件应用程序的回调方法。

- **void startDocument()** - 调用在一个文件的开头。

- **void endDocument()** - 调用在一个文件的末尾。
- **void startElement(String uri, String localName, String qName, Attributes atts)** - 调用在一个元素的开头。
- **void endElement(String uri, String localName,String qName)** - 调用在一个元件的末端。
- **void characters(char[] ch, int start, int length)** - 字符数据出现时调用。
- **void ignorableWhitespace( char[] ch, int start, int length)** - 当DTD是当前和忽略空白遇到时调用。
- **void processingInstruction(String target, String data)** - 当处理指令的认可时调用。
- **void setDocumentLocator(Locator locator))** - 提供可用于识别文档中的位置的定位器。
- **void skippedEntity(String name)** - 一个尚未解决实体遇到时调用。
- **void startPrefixMapping(String prefix, String uri)** - 当一个新的命名空间的映射定义调用。
- **void endPrefixMapping(String prefix)** - 当一个命名空间定义结束其范围时调用。

## 属性接口

这种接口指定用于处理连接到一个元素的属性的方法。

- **int getLength()** - 返回属性的数目。
- **String getQName(int index)**
- **String getValue(int index)**
- **String getValue(String qname)**

# Java SAX解析器 - 解析XML文档 - Java XML教程

## 演示示例

这是输入需要解析xml文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

*UserHandler.java*

```
package com.yiibai.xml;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class UserHandler extends DefaultHandler {

   boolean bFirstName = false;
   boolean bLastName = false;
   boolean bNickName = false;
   boolean bMarks = false;

   @Override
   public void startElement(String uri,
   String localName, String qName, Attributes attributes)
```

```
        throws SAXException {
        if (qName.equalsIgnoreCase("student")) {
           String rollNo = attributes.getValue("rollno");
           System.out.println("Roll No : " + rollNo);
        } else if (qName.equalsIgnoreCase("firstname")) {
           bFirstName = true;
        } else if (qName.equalsIgnoreCase("lastname")) {
           bLastName = true;
        } else if (qName.equalsIgnoreCase("nickname")) {
           bNickName = true;
        }
        else if (qName.equalsIgnoreCase("marks")) {
           bMarks = true;
        }
     }

     @Override
     public void endElement(String uri,
     String localName, String qName) throws SAXException {
        if (qName.equalsIgnoreCase("student")) {
           System.out.println("End Element :" + qName);
        }
     }

     @Override
     public void characters(char ch[],
        int start, int length) throws SAXException {
        if (bFirstName) {
           System.out.println("First Name: "
              + new String(ch, start, length));
           bFirstName = false;
        } else if (bLastName) {
           System.out.println("Last Name: "
              + new String(ch, start, length));
           bLastName = false;
        } else if (bNickName) {
           System.out.println("Nick Name: "
              + new String(ch, start, length));
           bNickName = false;
        } else if (bMarks) {
           System.out.println("Marks: "
              + new String(ch, start, length));
           bMarks = false;
        }
     }
  }
```

*SAXParserDemo.java*

```
package com.yiibai.xml;
```

```
import java.io.File;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SAXParserDemo {
   public static void main(String[] args){

      try {
         File inputFile = new File("input.txt");
         SAXParserFactory factory = SAXParserFactory.newInstance();
         SAXParser saxParser = factory.newSAXParser();
         UserHandler userhandler = new UserHandler();
         saxParser.parse(inputFile, userhandler);
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}

class UserHandler extends DefaultHandler {

   boolean bFirstName = false;
   boolean bLastName = false;
   boolean bNickName = false;
   boolean bMarks = false;

   @Override
   public void startElement(String uri,
      String localName, String qName, Attributes attributes)
         throws SAXException {
      if (qName.equalsIgnoreCase("student")) {
         String rollNo = attributes.getValue("rollno");
         System.out.println("Roll No : " + rollNo);
      } else if (qName.equalsIgnoreCase("firstname")) {
         bFirstName = true;
      } else if (qName.equalsIgnoreCase("lastname")) {
         bLastName = true;
      } else if (qName.equalsIgnoreCase("nickname")) {
         bNickName = true;
      }
      else if (qName.equalsIgnoreCase("marks")) {
         bMarks = true;
      }
   }

   @Override
   public void endElement(String uri,
      String localName, String qName) throws SAXException {
      if (qName.equalsIgnoreCase("student")) {
```

```
            System.out.println("End Element :" + qName);
         }
      }

      @Override
      public void characters(char ch[],
         int start, int length) throws SAXException {
         if (bFirstName) {
            System.out.println("First Name: "
            + new String(ch, start, length));
            bFirstName = false;
         } else if (bLastName) {
            System.out.println("Last Name: "
            + new String(ch, start, length));
            bLastName = false;
         } else if (bNickName) {
            System.out.println("Nick Name: "
            + new String(ch, start, length));
            bNickName = false;
         } else if (bMarks) {
            System.out.println("Marks: "
            + new String(ch, start, length));
            bMarks = false;
         }
      }
   }
```

这将产生以下结果：

```
Roll No : 393
First Name: dinkar
Last Name: kad
Nick Name: dinkar
Marks: 85
End Element :student
Roll No : 493
First Name: Vaneet
Last Name: Gupta
Nick Name: vinni
Marks: 95
End Element :student
Roll No : 593
First Name: jasvir
Last Name: singn
Nick Name: jazz
Marks: 90
End Element :student
```

# Java SAX解析器 - 查询XML文档 - Java XML教程

## 演示示例

这是我们需要查询卷号：rollno 输入文本文件：393

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

*UserHandler.java*

```
package com.yiibai.xml;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class UserHandler extends DefaultHandler {

   boolean bFirstName = false;
   boolean bLastName = false;
   boolean bNickName = false;
   boolean bMarks = false;
   String rollNo = null;

   @Override
   public void startElement(String uri,
```

```
    String localName, String qName, Attributes attributes)
    throws SAXException {

       if (qName.equalsIgnoreCase("student")) {
          rollNo = attributes.getValue("rollno");
       }
       if(("393").equals(rollNo) &&
          qName.equalsIgnoreCase("student")){
          System.out.println("Start Element :" + qName);
       }
       if (qName.equalsIgnoreCase("firstname")) {
          bFirstName = true;
       } else if (qName.equalsIgnoreCase("lastname")) {
          bLastName = true;
       } else if (qName.equalsIgnoreCase("nickname")) {
          bNickName = true;
       }
       else if (qName.equalsIgnoreCase("marks")) {
          bMarks = true;
       }
    }

    @Override
    public void endElement(String uri,
       String localName, String qName) throws SAXException {
       if (qName.equalsIgnoreCase("student")) {
          if(("393").equals(rollNo)
             && qName.equalsIgnoreCase("student"))
             System.out.println("End Element :" + qName);
       }
    }

    @Override
    public void characters(char ch[],
       int start, int length) throws SAXException {

       if (bFirstName && ("393").equals(rollNo)) {
          //age element, set Employee age
          System.out.println("First Name: " +
             new String(ch, start, length));
          bFirstName = false;
       } else if (bLastName && ("393").equals(rollNo)) {
          System.out.println("Last Name: " +
          new String(ch, start, length));
          bLastName = false;
       } else if (bNickName && ("393").equals(rollNo)) {
          System.out.println("Nick Name: " +
          new String(ch, start, length));
          bNickName = false;
       } else if (bMarks && ("393").equals(rollNo)) {
          System.out.println("Marks: " +
          new String(ch, start, length));
          bMarks = false;
```

```
      }
   }
}
```

*SAXQueryDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SAXQueryDemo {
   public static void main(String[] args){

      try {
         File inputFile = new File("input.txt");
         SAXParserFactory factory = SAXParserFactory.newInstance();
         SAXParser saxParser = factory.newSAXParser();
         UserHandler userhandler = new UserHandler();
         saxParser.parse(inputFile, userhandler);
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}

class UserHandler extends DefaultHandler {

   boolean bFirstName = false;
   boolean bLastName = false;
   boolean bNickName = false;
   boolean bMarks = false;
   String rollNo = null;

   @Override
   public void startElement(String uri,
      String localName, String qName, Attributes attributes)
      throws SAXException {

      if (qName.equalsIgnoreCase("student")) {
         rollNo = attributes.getValue("rollno");
      }
      if(("393").equals(rollNo) &&
         qName.equalsIgnoreCase("student")){
         System.out.println("Start Element :" + qName);
      }
```

```
         if (qName.equalsIgnoreCase("firstname")) {
            bFirstName = true;
         } else if (qName.equalsIgnoreCase("lastname")) {
            bLastName = true;
         } else if (qName.equalsIgnoreCase("nickname")) {
            bNickName = true;
         }
         else if (qName.equalsIgnoreCase("marks")) {
            bMarks = true;
         }
      }

      @Override
      public void endElement(String uri,
         String localName, String qName) throws SAXException {
         if (qName.equalsIgnoreCase("student")) {
            if(("393").equals(rollNo)
               && qName.equalsIgnoreCase("student"))
               System.out.println("End Element :" + qName);
            }
         }

      @Override
      public void characters(char ch[],
         int start, int length) throws SAXException {

         if (bFirstName && ("393").equals(rollNo)) {
            //age element, set Employee age
            System.out.println("First Name: " +
            new String(ch, start, length));
            bFirstName = false;
         } else if (bLastName && ("393").equals(rollNo)) {
            System.out.println("Last Name: " +
            new String(ch, start, length));
            bLastName = false;
         } else if (bNickName && ("393").equals(rollNo)) {
            System.out.println("Nick Name: " +
            new String(ch, start, length));
            bNickName = false;
         } else if (bMarks && ("393").equals(rollNo)) {
            System.out.println("Marks: " +
            new String(ch, start, length));
            bMarks = false;
         }
      }
   }
```

这将产生以下结果：

```
Start Element :student
First Name: dinkar
Last Name: kad
Nick Name: dinkar
Marks: 85
End Element :student
```

# Java SAX解析器 - 修改XML文档 - Java XML教程

## 演示示例

这是我们需要通过修改XML输入文件附加 **<Result>Pass<Result/>**

在**</marks>**标记结束

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

*SAXModifyDemo.java*

```
package com.yiibai.xml;

import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import org.xml.sax.helpers.DefaultHandler;

public class SAXModifyDemo extends DefaultHandler {
   static String displayText[] = new String[1000];
   static int numberLines = 0;
   static String indentation = "";

   public static void main(String args[]) {

      try {
```

```
         File inputFile = new File("input.txt");
         SAXParserFactory factory =
         SAXParserFactory.newInstance();
         SAXModifyDemo obj = new SAXModifyDemo();
            obj.childLoop(inputFile);
         FileWriter filewriter = new FileWriter("newfile.xml");
         for(int loopIndex = 0; loopIndex < numberLines; loopIndex-
            filewriter.write(displayText[loopIndex].toCharArray());
            filewriter.write('\n');
            System.out.println(displayText[loopIndex].toString());
         }
         filewriter.close();
      }
      catch (Exception e) {
         e.printStackTrace(System.err);
      }
   }

   public void childLoop(File input){
      DefaultHandler handler = this;
         SAXParserFactory factory = SAXParserFactory.newInstance();
      try {
         SAXParser saxParser = factory.newSAXParser();
         saxParser.parse(input, handler);
      } catch (Throwable t) {}
   }

   public void startDocument() {
      displayText[numberLines] = indentation;
      displayText[numberLines] += "<?xml version=\"1.0\" encoding=\
         "UTF-8" + "\"?>";
      numberLines++;
   }

   public void processingInstruction(String target,
      String data) {
      displayText[numberLines] = indentation;
      displayText[numberLines] += "<?";
      displayText[numberLines] += target;
      if (data != null && data.length() > 0) {
         displayText[numberLines] += ' ';
         displayText[numberLines] += data;
      }
      displayText[numberLines] += "?>";
      numberLines++;
   }

   public void startElement(String uri, String localName,
      String qualifiedName, Attributes attributes) {
      displayText[numberLines] = indentation;

      indentation += "    ";
```

```
        displayText[numberLines] += '<';
        displayText[numberLines] += qualifiedName;
        if (attributes != null) {
           int numberAttributes = attributes.getLength();
           for (int loopIndex = 0; loopIndex < numberAttributes;
              loopIndex++){
              displayText[numberLines] += ' ';
              displayText[numberLines] += attributes.getQName(loopInd
              displayText[numberLines] += "=\"";
              displayText[numberLines] += attributes.getValue(loopInd
              displayText[numberLines] += '"';
           }
        }
        displayText[numberLines] += '>';
        numberLines++;
     }

     public void characters(char characters[],
        int start, int length) {
        String characterData = (new String(characters, start, length)
        if(characterData.indexOf("\n") < 0 && characterData.length()
           displayText[numberLines] = indentation;
           displayText[numberLines] += characterData;
           numberLines++;
        }
     }

     public void endElement(String uri, String localName,
        String qualifiedName) {
        indentation = indentation.substring(0, indentation.length() -
        displayText[numberLines] = indentation;
        displayText[numberLines] += "</";
        displayText[numberLines] += qualifiedName;
        displayText[numberLines] += '>';
        numberLines++;

        if (qualifiedName.equals("marks")) {
              startElement("", "Result", "Result", null);
              characters("Pass".toCharArray(), 0, "Pass".length());
              endElement("", "Result", "Result");
        }
     }
  }
```

这将产生以下结果：

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
    <student rollno="393">
        <firstname>
```

```
            dinkar
        </firstname>
        <lastname>
            kad
        </lastname>
        <nickname>
            dinkar
        </nickname>
        <marks>
            85
        </marks>
        <Result>
            Pass
        </Result>
    </student>
    <student rollno="493">
        <firstname>
            Vaneet
        </firstname>
        <lastname>
            Gupta
        </lastname>
        <nickname>
            vinni
        </nickname>
        <marks>
            95
        </marks>
        <Result>
            Pass
        </Result>
    </student>
    <student rollno="593">
        <firstname>
            jasvir
        </firstname>
        <lastname>
            singn
        </lastname>
        <nickname>
            jazz
        </nickname>
        <marks>
            90
        </marks>
        <Result>
            Pass
        </Result>
    </student>
</class>
```

# Java JDOM解析器 - Java XML教程

JDOM是一个开源的，基于Java的库来解析XML文档，这是典型的Java开发人员友好的API。这是java的优化，它使用Java集合像列表和数组。它适用于DOM和SAX API并结合了两个中优点：低内存占用几乎和SAX一样快。

## 环境设置

为了使用JDOM解析器，应该有 jdom.jar 在应用程序的类路径中。下载 jdom-2.0.5.zip.

## 什么情况下使用？

应该使用JDOM解析器的情况：

- 需要知道很多关于文档的结构
- 需要将文档的部分围绕（例如，可能需要某些元素进行排序）
- 需要使用的文件中的信息超过一次
- Java开发人员，并希望利用Java的优化解析XML。

## 会得到什么？

当一个JDOM 解析器分析XML文档，可以灵活地得到一个树形结构，其中包含所有文档的元素，而不会影响应用程序的内存占用。JDOM提供了多种可用于检查的情况下的文件的文档的内容和结构的实用功能是良好的结构，其结构是公知的。

## 优势

JDOM使Java开发灵活性和XML解析代码易于维护。它是量轻级，快速API。

## JDOM 类

JDOM定义了几个Java类。以下是最常见的类：

- **Document** - 表示整个XML文档。文档Document对象是通常被称为DOM树。
- **Element** - 表示一个XML元素。 Element对象有方法来操作其子元素，它的文本，属性和名称空间。

- **Attribute** 表示元素的属性。属性有方法来获取和设置属性的值。它有家长和属性类型。
- **Text** 表示XML标记的文本。
- **Common** 表示一个XML文档中的注释。

## 常见的JDOM方法

使用JDOM，还有会经常用到的几种方法：

- **SAXBuilder.build(xmlSource)()** - 构建XML源的JDOM文档。
- **Document.getRootElement()** - 得到XML的根元素。
- **Element.getName()** - 获取XML节点的名称。
- **Element.getChildren()** - 得到一个元素的所有直接子节点。
- **Node.getChildren(Name)** - 获得具有给定名称的直接子节点。
- **Node.getChild(Name)** - 获取使用给定名称的第一个孩子节点。

# Java JDOM解析器 - 解析XML文档 - Java XML教程

## 使用JDOM的步骤

以下是解析时使用JDOM解析器文档的步骤。

- 导入XML相关的软件包
- 创建一个SAXBuilder
- 从文件或流创建一个文档
- 提取根元素
- 检查属性
- 检查子元素

导入XML相关的软件包

```
import java.io.*;
import java.util.*;
import org.jdom2.*;
```

创建**DocumentBuilder**

```
SAXBuilder saxBuilder = new SAXBuilder();
```

从文件或流创建一个文档

```
File inputFile = new File("input.txt");
SAXBuilder saxBuilder = new SAXBuilder();
Document document = saxBuilder.build(inputFile);
```

提取根元素

```
Element classElement = document.getRootElement();
```

检查属性

```
//returns specific attribute
getAttribute("attributeName");
```

检查子元素

```
//returns a list of subelements of specified name
getChildren("subelementName");
//returns a list of all child nodes
getChildren();
//returns first child node
getChild("subelementName");
```

## 演示示例

这是输入需要解析xml文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

演示示例：

*DomParserDemo.java*

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;

public class JDomParserDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");

         SAXBuilder saxBuilder = new SAXBuilder();

         Document document = saxBuilder.build(inputFile);

         System.out.println("Root element :"
            + document.getRootElement().getName());

         Element classElement = document.getRootElement();

         List<Element> studentList = classElement.getChildren();
         System.out.println("----------------------------");

         for (int temp = 0; temp < studentList.size(); temp++) {
            Element student = studentList.get(temp);
            System.out.println("\nCurrent Element :"
               + student.getName());
            Attribute attribute =  student.getAttribute("rollno");
            System.out.println("Student roll no : "
               + attribute.getValue() );
            System.out.println("First Name : " + student.getChild('
            System.out.println("Last Name : "+ student.getChild("la
            System.out.println("Nick Name : "+ student.getChild("ni
            System.out.println("Marks : "+ student.getChild("marks'
         }
      }catch(JDOMException e){
         e.printStackTrace();
      }catch(IOException ioe){
         ioe.printStackTrace();
      }
   }
}
```

这将产生以下结果：

```
Root element :class
----------------------------

Current Element :student
Student roll no : 393
First Name : dinkar
Last Name : kad
Nick Name : dinkar
Marks : 85

Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
Nick Name : vinni
Marks : 95

Current Element :student
Student roll no : 593
First Name : jasvir
Last Name : singn
Nick Name : jazz
Marks : 90
```

# Java JDOM解析器 - 查询XML文档 - Java XML教程

## 演示示例

这是需要我们查询的输入XML文件：

```
<?xml version="1.0"?>
<cars>
<supercars company="Ferrari">
<carname type="formula one">Ferarri 101</carname>
<carname type="sports car">Ferarri 201</carname>
<carname type="sports car">Ferarri 301</carname>
</supercars>
<supercars company="Lamborgini">
<carname>Lamborgini 001</carname>
<carname>Lamborgini 002</carname>
<carname>Lamborgini 003</carname>
</supercars>
<luxurycars company="Benteley">
<carname>Benteley 1</carname>
<carname>Benteley 2</carname>
<carname>Benteley 3</carname>
</luxurycars>
</cars>
```

演示示例：

*QueryXmlFileDemo.java*

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;

public class QueryXmlFileDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");

         SAXBuilder saxBuilder = new SAXBuilder();
```

```
         Document document = saxBuilder.build(inputFile);

         System.out.println("Root element :"
            + document.getRootElement().getName());

         Element classElement = document.getRootElement();

         List<Element> supercarList = classElement.getChildren("sup
         System.out.println("----------------------------");

         for (int temp = 0; temp < supercarList.size(); temp++) {
            Element supercarElement = supercarList.get(temp);
            System.out.println("\nCurrent Element :"
               + supercarElement.getName());
            Attribute attribute =  supercarElement.getAttribute("co
            System.out.println("company : "
               + attribute.getValue() );
            List<Element> carNameList = supercarElement.getChildren
            for (int count = 0;
               count < carNameList.size(); count++) {
               Element carElement = carNameList.get(count);
               System.out.print("car name : ");
               System.out.println(carElement.getText());
               System.out.print("car type : ");
               Attribute typeAttribute = carElement.getAttribute("t
               if(typeAttribute !=null)
                  System.out.println(typeAttribute.getValue());
               else{
                  System.out.println("");
               }
            }
         }
      }catch(JDOMException e){
         e.printStackTrace();
      }catch(IOException ioe){
         ioe.printStackTrace();
      }
   }
}
```

这将产生以下结果：

```
Root element :cars
----------------------------

Current Element :supercars
company : Ferrari
car name : Ferarri 101
car type : formula one
car name : Ferarri 201
car type : sports car
car name : Ferarri 301
car type : sports car

Current Element :supercars
company : Lamborgini
car name : Lamborgini 001
car type :
car name : Lamborgini 002
car type :
car name : Lamborgini 003
car type :
```

# Java JDOM解析器 - 创建XML文档 - Java XML教程

## 演示示例

这是我们需要创建XML：

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Ferrari">
    <carname type="formula one">Ferrari 101</carname>
    <carname type="sports">Ferrari 202</carname>
  </supercars>
</cars>
```

演示示例：

*CreateXmlFileDemo.java*

```
import java.io.IOException;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;

public class CreateXmlFileDemo {
   public static void main(String[] args) {

      try{
         //root element
         Element carsElement = new Element("cars");
         Document doc = new Document(carsElement);

         //supercars element
         Element supercarElement = new Element("supercars");
         supercarElement.setAttribute(new Attribute("company","Ferr

         //supercars element
         Element carElement1 = new Element("carname");
         carElement1.setAttribute(new Attribute("type","formula one
         carElement1.setText("Ferrari 101");

         Element carElement2 = new Element("carname");
         carElement2.setAttribute(new Attribute("type","sports"));
         carElement2.setText("Ferrari 202");

         supercarElement.addContent(carElement1);
         supercarElement.addContent(carElement2);

         doc.getRootElement().addContent(supercarElement);

         XMLOutputter xmlOutput = new XMLOutputter();

         // display ml
         xmlOutput.setFormat(Format.getPrettyFormat());
         xmlOutput.output(doc, System.out);
      }catch(IOException e){
         e.printStackTrace();
      }
   }
}
```

这将产生以下结果：

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Ferrari">
    <carname type="formula one">Ferrari 101</carname>
    <carname type="sports">Ferrari 202</carname>
  </supercars>
</cars>
```

# Java JDOM解析器 - 修改XML文档 - Java XML教程

## 演示示例

这是我们需要修改输入的文本文件：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars>
   <supercars company="Ferrari">
      <carname type="formula one">Ferrari 101</carname>
      <carname type="sports">Ferrari 202</carname>
   </supercars>
   <luxurycars company="Benteley">
      <carname>Benteley 1</carname>
      <carname>Benteley 2</carname>
      <carname>Benteley 3</carname>
   </luxurycars>
</cars>
```

演示示例：

*ModifyXmlFileDemo.java*

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;

public class ModifyXMLFileDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");
         SAXBuilder saxBuilder = new SAXBuilder();
         Document document = saxBuilder.build(inputFile);
         Element rootElement = document.getRootElement();

         //get first supercar
         Element supercarElement = rootElement.getChild("supercars'
```

```
         // update supercar attribute
         Attribute attribute = supercarElement.getAttribute("compar
         attribute.setValue("Lamborigini");

         // loop the supercar child node
         List<Element> list = supercarElement.getChildren();
         for (int temp = 0; temp < list.size(); temp++) {
            Element carElement = list.get(temp);
            if("Ferrari 101".equals(carElement.getText())){
               carElement.setText("Lamborigini 001");
            }
            if("Ferrari 202".equals(carElement.getText())){
               carElement.setText("Lamborigini 002");
            }
         }

         //get all supercars element
         List<Element> supercarslist = rootElement.getChildren();
         for (int temp = 0; temp < supercarslist.size(); temp++) {
            Element tempElement = supercarslist.get(temp);
            if("luxurycars".equals(tempElement.getName())){
               rootElement.removeContent(tempElement);
            }
         }

         XMLOutputter xmlOutput = new XMLOutputter();

         // display xml
         xmlOutput.setFormat(Format.getPrettyFormat());
         xmlOutput.output(document, System.out);
      } catch (JDOMException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

这将产生以下结果：

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Lamborigini">
    <carname type="formula one">Lamborigini 001</carname>
    <carname type="sports">Lamborigini 002</carname>
  </supercars>
</cars>
```

# Java StAX解析器 - Java XML教程

StAX是一个基于JAVA API用于解析XML文档，类似SAX解析器的方式。但两种API之间有两个区别

- StAX是PULL API，其中作为SAX是PUSH API。这意味着如果StAX解析器，客户端应用程序需要询问StAX解析器从XML获取信息它所需要的，但如果是SAX解析器，客户端应用程序需要获取信息时，SAX解析器会通知客户端应用程序的信息是可用的。
- StAX的API可以读取和写入XML文档。使用SAX API，XML可以是只读的。

## 环境设置

为了使用StAX的解析器，应该准备好stax.jar在应用程序的类路径中。下载 stax-1.2.0.jar.

以下是StAX API的功能

- 读取XML文件从上到下，认识构成一个结构完整的XML文档的标记
- 令牌是以相同的顺序进行处理，它们出现在文档中
- 报告应用程序，因为解析器遇到标记的特性
- 应用程序提供了一个“事件”读取器充当了事件，以获得所需信息的迭代器和迭代。可另一个读取器是“光标”充当一个指向XML节点。
- 由于事件被识别，XML元素可以从事件对象进行检索，并且可以进一步处理。

## 什么情况下使用？

应该使用的StAX解析器的时候：

- 可以处理在自上而下线性方式的XML文档。
- 文件并不深入嵌套。
- 处理一个非常大的XML文档的DOM树会占用太多的内存。典型的DOM的实现使用10字节的存储器以表示XML的一个字节。
- 要解决的问题涉及XML文档的一部分。
- 数据是可用的，只要它是由解析器处理，这样StAX可以很好地用于所收到超过数据流的XML文档。

## SAX的缺点

- 因为它是在一个处理的方式，而不是随机访问XML文档。
- 如果需要跟踪的数据分析器已经看到或更改项目的顺序，必须编写代码和数据存储以自己方式处理。

## XMLEventReader类

因为在解析XML文档时该类提供可用于迭代事件事件迭代器

- **StartElement asStartElement()** - 用于检索值和元素的属性。
- **EndElement asEndElement()** - 调用元件的端部。
- **Characters asCharacters()** - 可用于获得字符，例如一个CDATA，空白等。

## XMLEventWriter类

此接口指定创建事件的方法。

- **add(Event event)** - 添加包含元素XML事件。

## XMLStreamReader Class

因为在解析XML文档时该类提供可用于迭代事件事件迭代器

- **int next()** - 用于检索下一个事件。
- **boolean hasNext()** - 用于检查其他事件的存在与否
- **String getText()** - 用于获取一个元素的文本
- **String getLocalName()** - 用于获取一个元素的名称

## XMLStreamWriter类

此接口指定创建事件的方法

- **writeStartElement(String localName)** - 加入定名称开始元素。
- **writeEndElement(String localName)** - 添加指定名称的结束元素。
- **writeAttribute(String localName, String value)** - 编写属性到元素。

# Java StAX解析器 - 解析XML文档 - Java XML教程

## 演示示例

这是输入需要解析 xml 文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

*StAXParserDemo.java*

```
package com.yiibai.xml;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Iterator;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
```

```
public class StAXParserDemo {
   public static void main(String[] args) {
      boolean bFirstName = false;
      boolean bLastName = false;
      boolean bNickName = false;
      boolean bMarks = false;
      try {
         XMLInputFactory factory = XMLInputFactory.newInstance();
         XMLEventReader eventReader =
         factory.createXMLEventReader(
            new FileReader("input.txt"));

            while(eventReader.hasNext()){
               XMLEvent event = eventReader.nextEvent();
               switch(event.getEventType()){
                  case XMLStreamConstants.START_ELEMENT:
                     StartElement startElement = event.asStartEleme
                     String qName = startElement.getName().getLocal
                     if (qName.equalsIgnoreCase("student")) {
                        System.out.println("Start Element : student
                        Iterator<Attribute> attributes = startEleme
                        String rollNo = attributes.next().getValue(
                        System.out.println("Roll No : " + rollNo);
                     } else if (qName.equalsIgnoreCase("firstname")
                        bFirstName = true;
                     } else if (qName.equalsIgnoreCase("lastname"))
                        bLastName = true;
                     } else if (qName.equalsIgnoreCase("nickname"))
                        bNickName = true;
                     }
                     else if (qName.equalsIgnoreCase("marks")) {
                        bMarks = true;
                     }
                     break;
                  case XMLStreamConstants.CHARACTERS:
                     Characters characters = event.asCharacters();
                     if(bFirstName){
                        System.out.println("First Name: "
                        + characters.getData());
                        bFirstName = false;
                     }
                     if(bLastName){
                        System.out.println("Last Name: "
                        + characters.getData());
                        bLastName = false;
                     }
                     if(bNickName){
                        System.out.println("Nick Name: "
                        + characters.getData());
                        bNickName = false;
                     }
                     if(bMarks){
                        System.out.println("Marks: "
```

```
                          + characters.getData());
                          bMarks = false;
                       }
                       break;
                    case  XMLStreamConstants.END_ELEMENT:
                       EndElement endElement = event.asEndElement();
                       if(endElement.getName().getLocalPart().equalsI
                          System.out.println("End Element : student")
                          System.out.println();
                       }
                       break;
                 }
              }
           } catch (FileNotFoundException e) {
              e.printStackTrace();
           } catch (XMLStreamException e) {
              e.printStackTrace();
        }
     }
  }
```

这将产生以下结果：

```
Start Element : student
Roll No : 393
First Name: dinkar
Last Name: kad
Nick Name: dinkar
Marks: 85
End Element : student

Start Element : student
Roll No : 493
First Name: Vaneet
Last Name: Gupta
Nick Name: vinni
Marks: 95
End Element : student

Start Element : student
Roll No : 593
First Name: jasvir
Last Name: singn
Nick Name: jazz
Marks: 90
End Element : student
```

# Java StAX解析器 - 查询XML文档 - Java XML教程

## 演示示例

这是输入需要解析xml文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

*StAXParserDemo.java*

```
package com.yiibai.xml;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Iterator;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
```

```
public class StAXQueryDemo {
   public static void main(String[] args) {
      boolean bFirstName = false;
      boolean bLastName = false;
      boolean bNickName = false;
      boolean bMarks = false;
      boolean isRequestRollNo = false;
      try {
         XMLInputFactory factory = XMLInputFactory.newInstance();
         XMLEventReader eventReader =
         factory.createXMLEventReader(
            new FileReader("input.txt"));

         String requestedRollNo = "393";
         while(eventReader.hasNext()){
            XMLEvent event = eventReader.nextEvent();
            switch(event.getEventType()){
               case XMLStreamConstants.START_ELEMENT:
                  StartElement startElement = event.asStartElement(
                  String qName = startElement.getName().getLocalPar
                  if (qName.equalsIgnoreCase("student")) {
                     Iterator<Attribute> attributes = startElement.
                     String rollNo = attributes.next().getValue();
                     if(rollNo.equalsIgnoreCase(requestedRollNo)){
                        System.out.println("Start Element : student
                        System.out.println("Roll No : " + rollNo);
                        isRequestRollNo = true;
                     }
                  } else if (qName.equalsIgnoreCase("firstname")) {
                     bFirstName = true;
                  } else if (qName.equalsIgnoreCase("lastname")) {
                     bLastName = true;
                  } else if (qName.equalsIgnoreCase("nickname")) {
                     bNickName = true;
                  }
                  else if (qName.equalsIgnoreCase("marks")) {
                     bMarks = true;
                  }
                  break;
               case XMLStreamConstants.CHARACTERS:
                  Characters characters = event.asCharacters();
                  if(bFirstName && isRequestRollNo){
                     System.out.println("First Name: "
                     + characters.getData());
                     bFirstName = false;
                  }
                  if(bLastName && isRequestRollNo){
                     System.out.println("Last Name: "
                     + characters.getData());
                     bLastName = false;
                  }
                  if(bNickName && isRequestRollNo){
                     System.out.println("Nick Name: "
```

```
                        + characters.getData());
                        bNickName = false;
                     }
                     if(bMarks && isRequestRollNo){
                        System.out.println("Marks: "
                        + characters.getData());
                        bMarks = false;
                     }
                     break;
                  case  XMLStreamConstants.END_ELEMENT:
                     EndElement endElement = event.asEndElement();
                     if(endElement.getName().getLocalPart().equalsIgno
                        System.out.println("End Element : student");
                        System.out.println();
                        isRequestRollNo = false;
                     }
                  break;
               }
            }
         } catch (FileNotFoundException e) {
            e.printStackTrace();
         } catch (XMLStreamException e) {
            e.printStackTrace();
         }
      }
   }
```

这将产生以下结果:

```
Start Element : student
Roll No : 393
First Name: dinkar
Last Name: kad
Nick Name: dinkar
Marks: 85
End Element : student
```

# Java StAX解析器 - 创建XML文档 - Java XML教程

## 演示示例

这是我们需要创建的XML文档：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars><supercars company="Ferrari">
<carname type="formula one">Ferrari 101</carname>
<carname type="sports">Ferrari 202</carname>
</supercars></cars>
```

演示示例：

*StAXCreateXMLDemo.java*

```
package com.yiibai.xml;

import java.io.IOException;
import java.io.StringWriter;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

public class StAXCreateXMLDemo {
   public static void main(String[] args) {
      try {
         StringWriter stringWriter = new StringWriter();

         XMLOutputFactory xMLOutputFactory = XMLOutputFactory.newIn
         XMLStreamWriter xMLStreamWriter = xMLOutputFactory.createX

         xMLStreamWriter.writeStartDocument();
         xMLStreamWriter.writeStartElement("cars");

         xMLStreamWriter.writeStartElement("supercars");
         xMLStreamWriter.writeAttribute("company", "Ferrari");

         xMLStreamWriter.writeStartElement("carname");
         xMLStreamWriter.writeAttribute("type", "formula one");
         xMLStreamWriter.writeCharacters("Ferrari 101");
         xMLStreamWriter.writeEndElement();

         xMLStreamWriter.writeStartElement("carname");
         xMLStreamWriter.writeAttribute("type", "sports");
         xMLStreamWriter.writeCharacters("Ferrari 202");
```

```
            xMLStreamWriter.writeEndElement();

            xMLStreamWriter.writeEndElement();
            xMLStreamWriter.writeEndDocument();

            xMLStreamWriter.flush();
            xMLStreamWriter.close();

            String xmlString = stringWriter.getBuffer().toString();

            stringWriter.close();

            System.out.println(xmlString);

         } catch (XMLStreamException e) {
            e.printStackTrace();
         } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
         }
      }
   }
```

这将产生以下结果：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars><supercars company="Ferrari">
<carname type="formula one">Ferrari 101</carname>
<carname type="sports">Ferrari 202</carname>
</supercars></cars>
```

# Java StAX解析器 - 修改XML文档 - Java XML教程

## 演示示例

这是我们需要修改的XML文档：

```
 <class style="box-sizing: border-box;"><student rollno="393">
   <firstname>dinkar</firstname>
   <lastname>kad</lastname>
   <nickname>dinkar</nickname>
   <marks>85</marks>
</student>
<student rollno="493">
   <firstname>Vaneet</firstname>
   <lastname>Gupta</lastname>
   <nickname>vinni</nickname>
   <marks>95</marks>
</student>
<student rollno="593">
   <firstname>jasvir</firstname>
   <lastname>singh</lastname>
   <nickname>jazz</nickname>
   <marks>90</marks>
</student></class>
```

演示示例：

*StAXModifyDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
```

```
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;

public class StAXModifyDemo {
   public static void main(String[] args) {

      try {
         XMLInputFactory factory = XMLInputFactory.newInstance();
         XMLEventReader eventReader =
            factory.createXMLEventReader(
               new FileReader("input.txt"));
         SAXBuilder saxBuilder = new SAXBuilder();
         Document document = saxBuilder.build(new File("input.txt")
         Element rootElement = document.getRootElement();
         List<Element> studentElements = rootElement.getChildren("s
         while(eventReader.hasNext()){
            XMLEvent event = eventReader.nextEvent();
            switch(event.getEventType()){
               case XMLStreamConstants.START_ELEMENT:
               StartElement startElement = event.asStartElement();
               String qName = startElement.getName().getLocalPart(

               if (qName.equalsIgnoreCase("student")) {
                  Iterator<Attribute> attributes = startElement.get
                  String rollNo = attributes.next().getValue();
                  if(rollNo.equalsIgnoreCase("393")){
                     //get the student with roll no 393
                     for(int i=0;i < studentElements.size();i++){
                        Element studentElement = studentElements.ge
                        if(studentElement.getAttribute("rollno").ge
                           studentElement.removeChild("marks");
                           studentElement.addContent(new Element("r
                        }
                     }
                  }
               }
               break;
            }
         }
         XMLOutputter xmlOutput = new XMLOutputter();
         // display xml
         xmlOutput.setFormat(Format.getPrettyFormat());
         xmlOutput.output(document, System.out);
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (XMLStreamException e) {
         e.printStackTrace();
      } catch (JDOMException e) {
         e.printStackTrace();
```

```
        } catch (IOException e) {
           e.printStackTrace();
        }
     }
  }
```

这将产生以下结果:

```
<student rollno="393">
   <firstname>dinkar</firstname>
   <lastname>kad</lastname>
   <nickname>dinkar</nickname>
   <marks>80</marks>
</student>
<student rollno="493">
   <firstname>Vaneet</firstname>
   <lastname>Gupta</lastname>
   <nickname>vinni</nickname>
   <marks>95</marks>
</student>
<student rollno="593">
   <firstname>jasvir</firstname>
   <lastname>singh</lastname>
   <nickname>jazz</nickname>
   <marks>90</marks>
</student>
```

# Java XPath解析器 - Java XML教程

XPath是万维网联盟(W3C)的官方推荐。它定义了一个语言在XML文件中查找信息。它被用于遍历XML文档的元素和属性。 XPath提供各种类型，可用于从XML文档查询相关的信息表现形式。

## 什么是XPath？

- 结构定义 - XPath定义像元素，属性，文本，命名空间，处理指令，注释和文档节点的XML文档部分
- 路径表达式 - XPath提供了强大的路径表达式选择的节点或在XML文档中的节点列表。
- 标准功能 - XPath提供了丰富的标准函数库操纵字符串值，数值，日期和时间比较，节操作，顺序操作，布尔值等。
- **XSLT**重要组成部分 - XPath是在XSLT标准的主要元素之一，是必须有知识，以便使用XSLT的文档。
- **W3C**推荐 - XPath是万维网联盟(W3C)的官方推荐。

这里是我们需要分析输入文本文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

# XPath表达式

XPath使用路径表达式从XML文档中选择一个或多个节点的列表。以下是有用的路径和表达，从XML文档选择节点的任何节点/列表清单。

| 表达式 | 描述 |
|---|---|
| node-name | 选择具有给定名称的所有节点“nodename” |
| / | 选择从根节点开始 |
| // | 选择从当前节点匹配开始的选择 |
| . | 选择当前节点 |
| .. | 选择当前节点的父节点 |
| @ | 选择属性 |
| student | 例如：选择名称为“student”的所有节点 |
| class/student | 例如：选择属于类的所有学生的子类元素 |
| //student | 选择文档中所有学生的元素 |

## 谓词

谓词用于查找特定的节点或一个节点含有特定的值，并使用所定义 [...] .

| 表达式 | 结果 |
|---|---|
| /class/student[1] | 选择的是类元素的子第一个学生的元素 |
| /class/student[last()] | 选择的是类元素的子最后一个学生的元素 |
| /class/student[last()-1] | 选择的是类元素倒数的第二个学生的子元素 |
| //student[@rollno='493'] | 选择一个名为rollno为'493'值的属性的学生元素 |

# Java XPath解析器 - 解析XML文档 - Java XML教程

## 使用XPath的步骤

以下是使用XPath解析器在解析文档时使用的步骤。

- 导入XML相关的软件包。
- 创建DocumentBuilder
- 从文件或数据流创建一个文档
- 创建XPath对象和XPath的路径表达式
- 编译XPath表达式使用XPath.compile()， 并由XPath.evaluate()评估计算获得一个节点列表
- 遍历节点列表。
- 检查属性
- 检查子元素

导入**XML**相关的软件包

```
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
import java.io.*;
```

创建**DocumentBuilder**

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
```

从文件或数据流创建一个文档

```
StringBuilder xmlStringBuilder = new StringBuilder();
xmlStringBuilder.append("<?xml version="1.0"?> <class> </class>");
ByteArrayInputStream input =  new ByteArrayInputStream(
   xmlStringBuilder.toString().getBytes("UTF-8"));
Document doc = builder.parse(input);
```

构建**XPath**

```
XPath xPath =  XPathFactory.newInstance().newXPath();
```

准备路径表达式，并计算它

```
String expression = "/class/student";
NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(c
```

遍历节点列表

```
for (int i = 0; i < nodeList.getLength(); i++) {
   Node nNode = nodeList.item(i);
   ...
}
```

检查属性

```
//returns specific attribute
getAttribute("attributeName");
//returns a Map (table) of names/values
getAttributes();
```

检查子元素

```
//returns a list of subelements of specified name
getElementsByTagName("subelementName");
//returns a list of all child nodes
getChildNodes();
```

# 演示示例：

这里是我们需要分析输入文本文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singh</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

演示示例：

*XPathParserDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;

public class XPathParserDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");
         DocumentBuilderFactory dbFactory
            = DocumentBuilderFactory.newInstance();
```

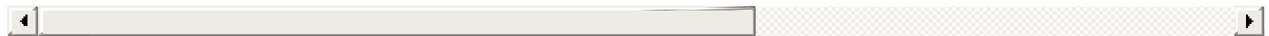
```
         DocumentBuilder dBuilder;

         dBuilder = dbFactory.newDocumentBuilder();

         Document doc = dBuilder.parse(inputFile);
         doc.getDocumentElement().normalize();

         XPath xPath =  XPathFactory.newInstance().newXPath();

         String expression = "/class/student";
         NodeList nodeList = (NodeList) xPath.compile(expression).e
         for (int i = 0; i < nodeList.getLength(); i++) {
            Node nNode = nodeList.item(i);
            System.out.println("\nCurrent Element :"
               + nNode.getNodeName());
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
               Element eElement = (Element) nNode;
               System.out.println("Student roll no : "
                  + eElement.getAttribute("rollno"));
               System.out.println("First Name : "
                  + eElement
                     .getElementsByTagName("firstname")
                     .item(0)
                     .getTextContent());
               System.out.println("Last Name : "
                  + eElement
                     .getElementsByTagName("lastname")
                     .item(0)
                     .getTextContent());
               System.out.println("Nick Name : "
                  + eElement
                     .getElementsByTagName("nickname")
                     .item(0)
                     .getTextContent());
               System.out.println("Marks : "
                  + eElement
                     .getElementsByTagName("marks")
                     .item(0)
                     .getTextContent());
            }
         }
      } catch (ParserConfigurationException e) {
         e.printStackTrace();
      } catch (SAXException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      } catch (XPathExpressionException e) {
         e.printStackTrace();
      }
   }
}
```



这将产生以下结果:

```
Current Element :student
Student roll no : 393
First Name : dinkar
Last Name : kad
Nick Name : dinkar
Marks : 85

Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
Nick Name : vinni
Marks : 95

Current Element :student
Student roll no : 593
First Name : jasvir
Last Name : singh
Nick Name : jazz
Marks : 90
```

# Java XPath解析器 - 查询XML文档 - Java XML教程

## 演示示例

这是我们需要查询输入的文本文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

演示示例：

*XPathParserDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
```

```
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;

public class XPathParserDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");
         DocumentBuilderFactory dbFactory
            = DocumentBuilderFactory.newInstance();
         DocumentBuilder dBuilder;

         dBuilder = dbFactory.newDocumentBuilder();

         Document doc = dBuilder.parse(inputFile);
         doc.getDocumentElement().normalize();

         XPath xPath =  XPathFactory.newInstance().newXPath();

         String expression = "/class/student[@rollno='493']";
         NodeList nodeList = (NodeList) xPath.compile(expression).e
         for (int i = 0; i < nodeList.getLength(); i++) {
            Node nNode = nodeList.item(i);
            System.out.println("\nCurrent Element :"
               + nNode.getNodeName());
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
               Element eElement = (Element) nNode;
               System.out.println("Student roll no : "
                  + eElement.getAttribute("rollno"));
               System.out.println("First Name : "
                  + eElement
                     .getElementsByTagName("firstname")
                     .item(0)
                     .getTextContent());
               System.out.println("Last Name : "
                  + eElement
                     .getElementsByTagName("lastname")
                     .item(0)
                     .getTextContent());
               System.out.println("Nick Name : "
                  + eElement
                     .getElementsByTagName("nickname")
                     .item(0)
                     .getTextContent());
               System.out.println("Marks : "
                  + eElement
                     .getElementsByTagName("marks")
                     .item(0)
                     .getTextContent());
            }
         }
```

```
        } catch (ParserConfigurationException e) {
           e.printStackTrace();
        } catch (SAXException e) {
           e.printStackTrace();
        } catch (IOException e) {
           e.printStackTrace();
        } catch (XPathExpressionException e) {
           e.printStackTrace();
        }
     }
  }
```

这将产生以下结果:

```
Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
Nick Name : vinni
Marks : 95
```

# Java DOM4J解析器 - Java XML教程

DOM4J是一个开源的，基于Java的库来解析XML文档，它具有高度的灵活性，高性能和内存效率的API。这是java的优化，使用Java集合像列表和数组。它可以使用DOM，SAX，XPath和XSLT。它解析大型XML文档时具有极低的内存占用。

## 环境设置

为了使用DOM4J解析器，应该 dom4j-1.6.1.jar 和 jaxen.jar 在应用程序的类路径中。下载 dom4j-1.6.1.zip.

## 什么情况下使用？

应该考虑使用DOM4J解析器的时候：

- 需要知道很多关于文档的结构
- 需要将文档的部分围绕（例如，可能需要某些元素进行排序）
- 需要使用的文件中的信息超过一次
- 你是一个Java开发人员，并希望利用XML的Java的优化解析。

## 会得到什么？

当解析一个DOM4J解析XML文档，可以灵活地得到一个树形结构，其中包含所有文档的元素，而不会影响应用程序的内存占用。DOM4J提供了多种可用于检查的情况下文档内容和结构的实用功能是良好的结构，其结构是公知的。 DOM4J使用XPath表达式来浏览XML文档。

## 优势

DOM4J使Java开发的灵活性和XML解析代码易于维护。它是轻量级的，快速的API。

## DOM4J 类

DOM4J定义了几个Java类。以下是最常见的类：

- **Document** - 表示整个XML文档。文档Document对象是通常被称为DOM树。

- **Element** - 表示一个XML元素。 Element对象有方法来操作其子元素，它的文本，属性和名称空间。
- **Attribute** - 表示元素的属性。属性有方法来获取和设置属性的值。它有父节点和属性类型。
- **Node** - 代表元素，属性或处理指令

## 常见DOM4J的方法

当使用DOM4J，还有经常用到的几种方法：

- **SAXReader.read(xmlSource)()** - 构建XML源的DOM4J文档。
- **Document.getRootElement()** - 得到的XML的根元素。
- **Element.node(index)** - 获得在元素特定索引XML节点。
- **Element.attributes()** - 获取一个元素的所有属性。
- **Node.valueOf(@Name)** - 得到元件的给定名称的属性的值。

# Java DOM4J解析器 - 解析XML文档 - Java XML教程

## 使用DOM4J的步骤

以下是解析时使用DOM4J解析器文档使用的步骤。

- 导入XML相关的软件包。
- 创建一个SAXReader
- 从文件或数据流创建一个文档
- 通过调用document.selectNodes()获取使用XPath表达式所需的节点
- 提取根元素
- 遍历节点列表。
- 检查属性
- 检查子元素

导入**XML**相关的软件包

```
import java.io.*;
import java.util.*;
import org.dom4j.*;
```

创建一个**DocumentBuilder**

```
SAXBuilder saxBuilder = new SAXBuilder();
```

从文件或数据流创建一个文档

```
File inputFile = new File("input.txt");
SAXBuilder saxBuilder = new SAXBuilder();
Document document = saxBuilder.build(inputFile);
```

提取根元素

```
Element classElement = document.getRootElement();
```

### Examine attributes

```
//returns specific attribute
valueOf("@attributeName");
```

检查子元素

```
//returns first child node
selectSingleNode("subelementName");
```

# 演示示例

这是输入需要解析xml文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

演示示例：

*DOM4JParserDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import java.util.List;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.SAXReader;

public class DOM4JParserDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");
         SAXReader reader = new SAXReader();
         Document document = reader.read( inputFile );

         System.out.println("Root element :"
            + document.getRootElement().getName());

         Element classElement = document.getRootElement();

         List<Node> nodes = document.selectNodes("/class/student" )
         System.out.println("----------------------------");
         for (Node node : nodes) {
            System.out.println("\nCurrent Element :"
               + node.getName());
            System.out.println("Student roll no : "
               + node.valueOf("@rollno") );
            System.out.println("First Name : " + node.selectSingleN
            System.out.println("Last Name : " + node.selectSingleNo
            System.out.println("First Name : " + node.selectSingleN
            System.out.println("Marks : " + node.selectSingleNode('
         }
      } catch (DocumentException e) {
         e.printStackTrace();
      }
   }
}
```

这将产生以下结果:

```
Root element :class
----------------------------

Current Element :student
Student roll no :
First Name : dinkar
Last Name : kad
First Name : dinkar
Marks : 85

Current Element :student
Student roll no :
First Name : Vaneet
Last Name : Gupta
First Name : vinni
Marks : 95

Current Element :student
Student roll no :
First Name : jasvir
Last Name : singn
First Name : jazz
Marks : 90
```

# Java DOM4J解析器 - 查询XML文档 - Java XML教程

## 演示示例

这是输入需要解析xml文件：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

演示示例：

*DOM4JQueryDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import java.util.List;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.SAXReader;

public class DOM4JQueryDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");
         SAXReader reader = new SAXReader();
         Document document = reader.read( inputFile );

         System.out.println("Root element :"
            + document.getRootElement().getName());

         Element classElement = document.getRootElement();

         List<Node> nodes = document.selectNodes("/class/student[@r
         System.out.println("----------------------------");
         for (Node node : nodes) {
            System.out.println("\nCurrent Element :"
               + node.getName());
            System.out.println("Student roll no : "
               + node.valueOf("@rollno") );
            System.out.println("First Name : " + node.selectSingleN
            System.out.println("Last Name : " + node.selectSingleNo
            System.out.println("First Name : " + node.selectSingleN
            System.out.println("Marks : " + node.selectSingleNode('
         }
      } catch (DocumentException e) {
         e.printStackTrace();
      }
   }
}
```

这将产生以下结果:

```
Root element :class
----------------------------
Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
First Name : vinni
Marks : 95
```

# Java DOM4J解析器 - 创建XML文档 - Java XML教程

## 演示示例

这是我们需要创建的XML文档：

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Ferrari">
    <carname type="formula one">Ferrari 101</carname>
    <carname type="sports">Ferrari 202</carname>
  </supercars>
</cars>
```

演示示例：

*CreateXmlFileDemo.java*

```
package com.yiibai.xml;

import java.io.IOException;
import java.io.UnsupportedEncodingException;

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.XMLWriter;

public class DOM4JCreateXMLDemo {
   public static void main(String[] args) {
      try {
         Document document = DocumentHelper.createDocument();
         Element root = document.addElement( "cars" );
         Element supercarElement= root.addElement("supercars")
            .addAttribute("company", "Ferrai");

         supercarElement.addElement("carname")
            .addAttribute("type", "Ferrari 101")
            .addText("Ferrari 101");

         supercarElement.addElement("carname")
            .addAttribute("type", "sports")
            .addText("Ferrari 202");

         // Pretty print the document to System.out
         OutputFormat format = OutputFormat.createPrettyPrint();
         XMLWriter writer;
         writer = new XMLWriter( System.out, format );
         writer.write( document );
      } catch (UnsupportedEncodingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

这将产生以下结果:

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Ferrari">
    <carname type="formula one">Ferrari 101</carname>
    <carname type="sports">Ferrari 202</carname>
  </supercars>
</cars>
```

# Java DOM4J解析器 - 修改XML文档 - Java XML教程

## 演示示例

这是我们需要修改的XML文档：

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

演示示例：

*DOM4jModifyXMLDemo.java*

```
package com.yiibai.xml;

import java.io.File;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.Iterator;
import java.util.List;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.SAXReader;
import org.dom4j.io.XMLWriter;

public class DOM4jModifyXMLDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");
         SAXReader reader = new SAXReader();
         Document document = reader.read( inputFile );

         Element classElement = document.getRootElement();

         List<Node> nodes = document.selectNodes("/class/student[@r
         for (Node node : nodes) {
            Element element = (Element)node;
            Iterator<Element> iterator=element.elementIterator("mar
            while(iterator.hasNext()){
               Element marksElement=(Element)iterator.next();
               marksElement.setText("80");
            }
         }

         // Pretty print the document to System.out
         OutputFormat format = OutputFormat.createPrettyPrint();
         XMLWriter writer;
         writer = new XMLWriter( System.out, format );
         writer.write( document );
      } catch (DocumentException e) {
         e.printStackTrace();
      } catch (UnsupportedEncodingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

这将产生以下结果:

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
  <student rollno="393">
    <firstname>dinkar</firstname>
    <lastname>kad</lastname>
    <nickname>dinkar</nickname>
    <marks>85</marks>
  </student>
  <student rollno="493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>vinni</nickname>
    <marks>80</marks>
  </student>
  <student rollno="593">
    <firstname>jasvir</firstname>
    <lastname>singn</lastname>
    <nickname>jazz</nickname>
    <marks>90</marks>
  </student>
</class>
```

# Java XML学习资源 - Java XML教程

## XML语法分析网站

- JDOM - 专门为软件开发人员和使用JDOM XML解析库测试。
- JDOM - 专门为软件开发人员和使用DOM4J XML解析库测试。
- StAX - 专门为软件开发人员和使用StAX的XML解析库测试。
- Java2 SDK，标准版 - 官方网站的Java2 SDK，标准版
- Java下载 - 下载Java！
- Java XML教程 - Java XML教程中文版本！

# JavaMail API 教程

JavaMail API提供了一种与平台无关和协议独立的框架来构建邮件和消息应用程序。 JavaMail API提供了一组抽象类定义构成一个邮件系统的对象。它是阅读，撰写和发送电子信息的可选包（标准扩展）。

## 读者

本教程乃为初学者，帮助他们了解基本的JavaMail的编程。完成本教程后，在一定程度上提高了你自己的 JavaMail 的编程水平。

## 必备条件

JavaMail 编程是基于Java 编程语言，所以如果你对Java编程有基本的了解，那么在应用程序开发中使用JavaMail 会比较顺手。

# JavaMail API 概述 - JavaMail

JavaMail API提供了一种与平台无关和协议独立的框架来构建邮件和消息应用程序。 JavaMail API提供了一组抽象类定义构成一个邮件系统的对象。它是阅读，撰写和发送电子信息的可选包（标准扩展）。

JavaMail 规定，用于构造一个接口，一个消息传送系统中的元素，包括系统的部件和接口。虽然本规范没有定义任何特定的实现，JavaMail是否包括实现RFC822和MIME Internet邮件标准几类。这些类都作为JavaMail的类包的一部分。

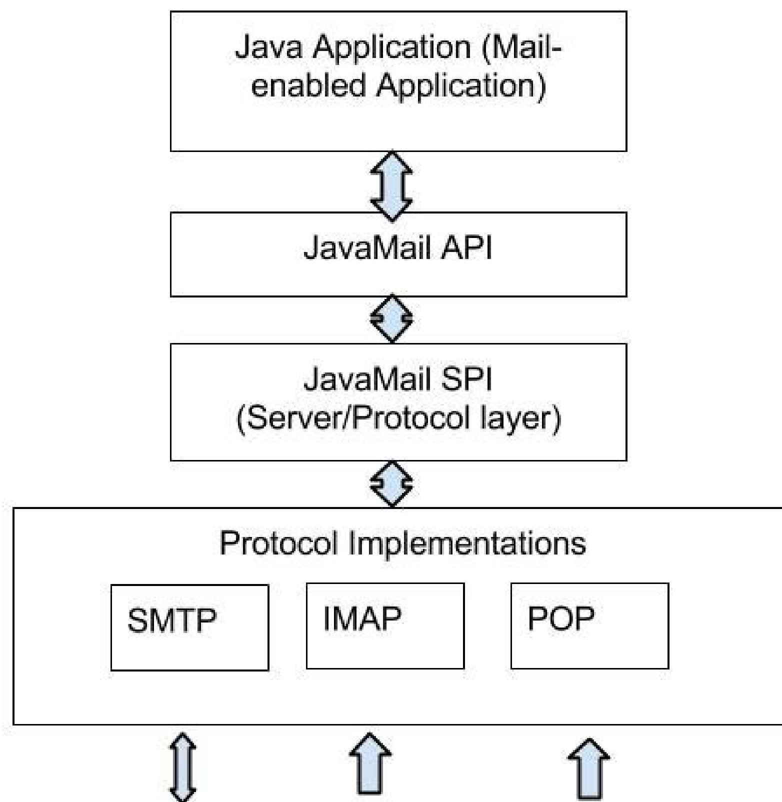
以下是一些在 JavaMail API 支持的协议：

- SMTP:缩写为简单邮件传输协议。它提供传送邮件的机制。
- POP: 缩写为邮局协议。 POP是大多数人在互联网上使用，以获得他们的邮件的机制。它定义了一个单个邮箱的支持为每个用户。 RFC 1939定义了该协议。
- IMAP: 缩写为Internet邮件访问协议。它是一种先进的协议，用于接收消息。它提供了多个邮箱的支持为每个用户，除了邮箱可以被多个用户共享。它是在RFC2060中定义。
- MIME: 缩写为多用途Internet邮件扩展。 。这不是一个邮件传输协议。相反，它定义了什么是传输的内容：邮件，附件，等等的格式。有许多不同的文档生效这里：RFC822，RFC2045，RFC2046和RFC2047。作为 JavaMail API 用户，您通常不需要担心这些格式。然而，这些格式确实存在，并且由程序使用。
- NNTP 其它: 有由第三方供应商提供的许多协议。其中有些是网络新闻传输协议（NNTP），安全多用途Internet邮件扩展（S / MIME）等。

这些细节将包括在后续章节。

## 体系结构

正如上面的 java 应用程序表示用户使用 JavaMail API 来编写，发送和接收电子邮件。下图说明了JavaMail 的体系结构：




JavaMail API 的抽象机制类似于其他的J2EE API，如JDBC，JNDI和JMS。如上面看到的体系结构图，JavaMail API 被分为两个主要部分：

- 与应用程序无关的部分：应用程序编程接口（API）是由应用程序使用的组件来发送和接收邮件，独立于底层的供应商或协议中使用的。
- 一个服务相关的部分：一个服务提供商接口（SPI）说，协议特定的语言，如SMTP，POP，IMAP和网络新闻传输协议（NNTP）。它是用来插在一封邮件服务到J2EE平台的供应商。

# JavaMail API 环境设置 - JavaMail

要使用Java应用程序发送邮件是很简单的，但首先应该有安装 JavaMail API 和 Java 激活框架（JAF）。

> 您需要JavaBeans激活框架（JAF）的扩展，当你不使用Java SE6或更高版本提供了javax.activation中的包。

- 您可以从Java 标准网站下载最新版本的 JavaMail（版本1.5.0）。
- 您可以从Java的标准网站下载最新版本的 JAF（版本1.1.1）。

下载并解压缩这些文件，在新创建的顶层目录，你会发现一些jar文件同时为应用。需要添加 mail.jar 和 activation.jar 文件在CLASSPATH中。

## SMPT 服务器

发送电子邮件，您必须有SMTP服务器，它负责发送邮件。您可以使用下列方法之一来获取SMTP服务器：

- 安装和使用任何SMTP服务器，如 Postfix 服务器（Ubuntu），James Apache 服务器（Apache 的 Java 的企业邮件服务器）等。
- 使用由主机供应商如提供的SMTP服务器：免费 SMTP 通过JangoSMTP 网站提供的是 relay.jangosmtp.net。
- 使用由公司提供的SMTP服务器如Gmail，雅虎等。

> 在随后的章节中的例子中，我们使用了免费 JangoSMTP 服务器发送电子邮件。您可以通过访问这个网站上创建一个帐户，并配置您的电子邮件地址。

# JavaMail API 核心类 - JavaMail

JavaMail API包含了一些接口，用于发送，读取和删除电子邮件消息的类。虽然有许多软件包在JavaMail API中，频繁用于Java邮件API主要有两个包：javax.mail和javax.mail.internet。这些软件包包含所有的JavaMail核心类。它们分别是：

| 类 | 描述 |
|---|---|
| javax.mail.Session | The key class of the API. A multithreaded object represents the connection factory. |
| javax.mail.Message | An abstract class that models an e-mail message. Subclasses provide the actual implementations. |
| javax.mail.Address | An abstract class that models the addresses (from and to addresses) in a message. Subclasses provide the specific implementations. |
| javax.mail.Authenticator | An abstract class used to protect mail resources on the mail server. |
| javax.mail.Transport | An abstract class that models a message transport mechanism for sending an e-mail message. |
| javax.mail.Store | An abstract class that models a message store and its access protocol, for storing and retrieving messages. A Store is divided into Folders. |
| javax.mail.Folder | An abstract class that represents a folder of mail messages. It can contain subfolders. |
| javax.mail.internet.**MimeMessage** | Message is an abstract class, hence must work with a subclass; in most cases, you'll use a MimeMessage. A MimeMessage is an e-mail message that understands MIME types and headers. |
| javax.mail.internet.**InternetAddress** | This class represents an Internet email address using the syntax of RFC822. Typical address syntax is of the form*user@host.domain* or *Personal Name <user@host.domain>*. |

让我们研究这些类的细节，并在随后的章节中，我们将使用所有这些研究的例子。

## 会话类

会话类是JavaMail API的主要类，它不创建子类。 Session 对象充当连接工厂的 JavaMail API，它可以同时处理配置设置和身份验证。

Session 对象可以通过以下方式创建：

- 通过查找存储在JNDI服务的管理对象

  ```
  InitialContext ctx = new InitialContext();
  Session session = (Session) ctx.lookup("usersMailSession");
  ```

  usersMailSession是用作Session 对象的管理对象的JNDI名称的对象。usersMailSession 可以创建并配置必要的参数作为名称/值对，包括信息，如邮件服务器的主机名，用户帐户发送邮件，并通过 Session 对象所支持的协议。

- 创建Session对象的另一种方法是基于编程方法，可以在其中使用的java.util.Properties对象来覆盖一些默认信息，如邮件服务器名，用户名，密码，那可以是其他信息整个应用程序共享。

该构造Session类是私有的。因此，会话类提供了两个方法（如下所示），它获得了Session对象。

- getDefaultInstance(): 有两种方法使用getDefaultInstance（）方法来获取会话对象。它返回默认的会话。

  ```
  public static Session getDefaultInstance(Properties props)
  public static Session getDefaultInstance(Properties props,Authe
  ```

- **getInstance()**: 有两种方法使用getInstance（）方法来获取会话对象。它返回新的会话。

  ```
  public static Session getInstance(Properties props)
  public static Session getInstance(Properties props,Authenticato
  ```

## 消息类

与Session对象创建的，我们现在继续创建将要发送的消息。该消息类型将是javax.mail.Message。

- Message是一个抽象类。因此，它的子类javax.mail.internet.MimeMessage类大多使用。

- 创建消息，你需要传递会话对象中的MimeMessage类的构造函数。例如：

  ```
  MimeMessage message=new MimeMessage(session);
  ```

- 一旦消息对象被创建，我们需要存储的信息在里面。消息类实现了javax.mail.Part接口，当使用javax.mail.internet。的MimeMessage实现javax.mail.internet.MimePart。您可以使用message.setContent()或mimeMessage.setText()来存储内容。

- MimeMessage类的常用的方法有

| 方法 | 描述 |
|---|---|
| public void setFrom(Address address) | used to set the from header field. |
| public void addRecipients(Message.RecipientType type, String addresses) | used to add the given address to the recipient type. |
| public void setSubject(String subject) | used to set the subject header field. |
| public void setText(String textmessage) | used to set the text as the message content using text/plain MIME type. |

## 地址类

现在，我们有一个会话和消息（存储在它里面的内容）的对象，我们需要使用地址对象，以解决这封邮件。

- Address 是一个抽象类。因此，它的子类javax.mail.internet.InternetAddress类大多被使用。

- Address 可以通过刚好路过的电子邮件地址来创建：

```
Address address = new InternetAddress("manisha@gmail.com");
```

- 创建地址的另一种方式是通过将名称与电子邮件地址：

```
Address address = new InternetAddress("manisha@gmail.com", Mani
```

- 您还可以设置收件人，发件人，抄送，密件抄送（To, From, CC, BCC）字段如下 fields as below
    - message.setFrom(address)
    - message.addRecipient(type, address)
    - 三种预定义的地址类型是与这些值中的一个对象：
        - Message.RecipientType.TO
        - Message.RecipientType.CC

- Message.RecipientType.BCC

## Authenticator类

Authenticator 类表示懂得如何获得认证的网络连接的对象。通常情况下，它会通过提示信息的用户这样做。

- 身份验证是一个抽象类。您可以创建一个子类PasswordAuthentication，通过用户名和密码给它的构造。
- 必须注册认证者与当您创建会话对象的会话。

以下是验证器使用的一个例子：

```
Properties props = new Properties();
//Override props with any customized data
PasswordAuthentication auth = new PasswordAuthentication("manisha",
Session session = Session.getDefaultInstance(props, auth);
```

## Transport 类

Transport 类用来作为消息传输机制。这个类通常使用SMTP协议来发送消息。

- 它是一个抽象类。
- 你可以通过只调用静态的send（）方法使用该类的默认版本：

```
Transport.send(message);
```

- 发送消息的另一种方法是通过从会话您的协议得到一个特定的实例，传递下去的用户名和密码（空白，如果不必要的），发送消息，并关闭连接：

```
message.saveChanges(); // implicit with send()
//Get transport for session
Transport transport = session.getTransport("smtp");
//Connect
transport.connect(host, username, password);
//repeat if necessary
transport.sendMessage(message, message.getAllRecipients());
//Done, close the connection
transport.close();
```

## Store 类

一个抽象类，模型信息存储和访问协议，用于存储和检索信息。子类提供实际的实现。存储扩展服务类，它提供命名商店，连接到存储，并听取连接事件很多常见的方法。

客户获得通过获得它实现了数据库访问协议的Store对象访问消息存储。大多数邮件存储需要进行身份验证，才允许访问的用户。 connect方法进行身份验证。

```
Store store = session.getStore("pop3");
store.connect(host, username, password);
```

## Folder 类

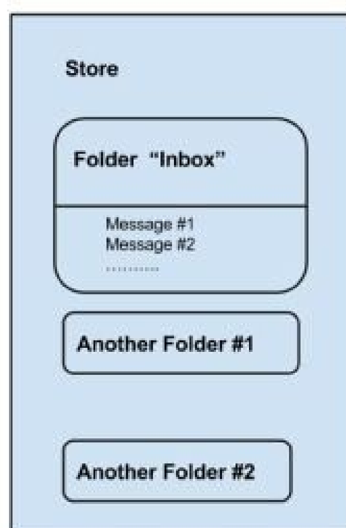
folder 是表示一个文件夹的邮件消息的抽象类。子类实现协议的具体文件夹。文件夹可以包含子文件夹，以及消息，从而提供了一种分层结构。

连接到存储后，您就可以得到一个文件夹，必须先打开，然后才能从中读取消息。

```
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
Message message[] = folder.getMessages();
```

getFolder（字符串name）方法为一个Folder 对象返回指定的子文件夹。关闭一次读邮件完成两者的存储和文件夹的连接。

我们可以看到下面的图片的存储和文件夹的关系：



正如我们所看到的，每个用户帐户，该服务器有一个商店，这是用户的信息的存储。该存储分为文件夹，并在“收件箱”文件夹，其中包含电子邮件的主要文件夹。文件夹可以包含邮件和子文件夹。

# JavaMail API 发送电子邮件 - JavaMail

现在，我们对JavaMail API及其核心类有一个清晰的概念，现在让我们写这将发送简单的电子邮件，邮件带有附件，电子邮件，HTML内容和电子邮件内嵌图像一个简单的程序。

接着在上述所有情况的基本步骤如下：

- 获取Session对象。
- 撰写邮件。
- 发送消息。

在下面的章节中，我们已经证明了简单的例子：

- 发送简单邮件
- 发送附件的邮件
- 在电子邮件中发送HTML内容
- 发送内嵌图像中的电子邮件

# JavaMail 查询电子邮件 - JavaMail

分为两部分，需要在继续本章之前的理解。邮件检查和提取。

- 检查在JavaMail的电子邮件是我们在邮箱打开相应的文件夹，每个消息传递的过程。在这里，我们只检查每封邮件的标题即发件人，收件人，主题。内容不会被读取。
- 在获取JavaMail的电子邮件是我们在邮箱中打开相应的文件夹，每个消息传递的过程。非常久远的标题，我们也认识到内容类型阅读的内容。

为了使用JavaMail API 检查或提取电子邮件，我们需要 POP或IMAP服务器。要检查并获取邮件，需要文件夹和存储类。在这里，我们使用了Gmail 的 POP3服务器（pop.gmail.com）。在本章将学习如何使用JavaMail API 来检查电子邮件。提取应覆盖在随后的章节。要检查的电子邮件：

- 获得一个 Session
- 创建POP3 Store对象并连接pop服务器。
- 创建文件夹对象。在您的邮箱中打开相应的文件夹。
- 得到消息。
- 关闭存储和文件夹对象。

## 创建Java类

创建一个Java类文件CheckingMails，是其内容如下：

```
package com.yiibai;

import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.Session;
import javax.mail.Store;

public class CheckingMails {

   public static void check(String host, String storeType, String u
      String password)
   {
      try {
```

```
        //create properties field
        Properties properties = new Properties();

        properties.put("mail.pop3.host", host);
        properties.put("mail.pop3.port", "995");
        properties.put("mail.pop3.starttls.enable", "true");
        Session emailSession = Session.getDefaultInstance(properties)

        //create the POP3 store object and connect with the pop serve
        Store store = emailSession.getStore("pop3s");

        store.connect(host, user, password);

        //create the folder object and open it
        Folder emailFolder = store.getFolder("INBOX");
        emailFolder.open(Folder.READ_ONLY);

        // retrieve the messages from the folder in an array and prin
        Message[] messages = emailFolder.getMessages();
        System.out.println("messages.length---" + messages.length);

        for (int i = 0, n = messages.length; i < n; i++) {
           Message message = messages[i];
           System.out.println("---------------------------------");
           System.out.println("Email Number " + (i + 1));
           System.out.println("Subject: " + message.getSubject());
           System.out.println("From: " + message.getFrom()[0]);
           System.out.println("Text: " + message.getContent().toStrin

        }

        //close the store and folder objects
        emailFolder.close(false);
        store.close();

        } catch (NoSuchProviderException e) {
           e.printStackTrace();
        } catch (MessagingException e) {
           e.printStackTrace();
        } catch (Exception e) {
           e.printStackTrace();
        }
     }

     public static void main(String[] args) {

        String host = "pop.gmail.com";// change accordingly
        String mailStoreType = "pop3";
        String username = "yourmail@gmail.com";// change accordingly
        String password = "*****";// change accordingly

        check(host, mailStoreType, username, password);

```

```
    }

}
```

# 编译并运行

现在，我们班是准备好了，让我们编译上面的类。我已经保存了类 CheckingMails.java目录： /home/manisha/JavaMailAPIExercise. 我们需要 javax.mail.jar andactivation.jar 在classpath中。执行下面的命令从命令提示符编译类（两个jar文件被放置在 /home/manisha/ 目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

# 验证输出

你应该看到下面的消息命令控制台上：

```
messages.length---4
---------------------------------
Email Number 1
Subject: Test Mail--Fetch
From: <abcd@gmail.com>
Text: javax.mail.internet.MimeMultipart@327a5b7f
---------------------------------
Email Number 2
Subject: testing ----checking simple email
From: <abcd@gmail.com>
Text: javax.mail.internet.MimeMultipart@7f0d08bc
---------------------------------
Email Number 3
Subject: Email with attachment
From: <abcd@gmail.com>
Text: javax.mail.internet.MimeMultipart@30b8afce
---------------------------------
Email Number 4
Subject: Email with Inline image
From: <abcd@gmail.com>
Text: javax.mail.internet.MimeMultipart@2d1e165f
```

这里，我们已经打印的消息，其中是4在这种情况下，收件箱的数目。我们还印制主题，发件人地址和文本的每封电子邮件。

# JavaMail 获取电子邮件 - JavaMail

在前面的章节中，我们学会了如何检查电子邮件。现在让我们来看看如何获取每封电子邮件和阅读其内容。让我们编写一个Java类FetchingEmail 读取以下类型的电子邮件：

- 简单的电子邮件
- 电子邮件与附件
- 电子邮件与内嵌图像

其次在代码的基本步骤如下：

- 获取Session对象。
- 创建POP3存储对象，并连接到存储。
- 创建文件夹对象，并在您的邮箱中打开相应的文件夹。
- 检索消息。
- 分别关闭文件夹和存储对象。

## 创建Java类

创建一个Java创建一个Java类文件FetchingEmail，内容都是如下：

```
package com.yiibai;

import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.Properties;

import javax.mail.Address;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.NoSuchProviderException;
import javax.mail.Part;
import javax.mail.Session;
```

```
import javax.mail.Store;

public class FetchingEmail {

   public static void fetch(String pop3Host, String storeType, Stri
      String password) {
      try {
         // create properties field
         Properties properties = new Properties();
         properties.put("mail.store.protocol", "pop3");
         properties.put("mail.pop3.host", pop3Host);
         properties.put("mail.pop3.port", "995");
         properties.put("mail.pop3.starttls.enable", "true");
         Session emailSession = Session.getDefaultInstance(properti
         // emailSession.setDebug(true);

         // create the POP3 store object and connect with the pop s
         Store store = emailSession.getStore("pop3s");

         store.connect(pop3Host, user, password);

         // create the folder object and open it
         Folder emailFolder = store.getFolder("INBOX");
         emailFolder.open(Folder.READ_ONLY);

         BufferedReader reader = new BufferedReader(new InputStrean
          System.in));

         // retrieve the messages from the folder in an array and p
         Message[] messages = emailFolder.getMessages();
         System.out.println("messages.length---" + messages.length)

         for (int i = 0; i < messages.length; i++) {
            Message message = messages[i];
            System.out.println("---------------------------------")
            writePart(message);
            String line = reader.readLine();
            if ("YES".equals(line)) {
               message.writeTo(System.out);
            } else if ("QUIT".equals(line)) {
               break;
            }
         }

         // close the store and folder objects
         emailFolder.close(false);
         store.close();

      } catch (NoSuchProviderException e) {
         e.printStackTrace();
      } catch (MessagingException e) {
         e.printStackTrace();
      } catch (IOException e) {
```

```
            e.printStackTrace();
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
      public static void main(String[] args) {

         String host = "pop.gmail.com";// change accordingly
         String mailStoreType = "pop3";
         String username =
            "abc@gmail.com";// change accordingly
         String password = "*****";// change accordingly

         //Call method fetch
         fetch(host, mailStoreType, username, password);

      }

      /*
      * This method checks for content-type
      * based on which, it processes and
      * fetches the content of the message
      */
      public static void writePart(Part p) throws Exception {
         if (p instanceof Message)
            //Call methos writeEnvelope
            writeEnvelope((Message) p);

         System.out.println("----------------------------");
         System.out.println("CONTENT-TYPE: " + p.getContentType());

         //check if the content is plain text
         if (p.isMimeType("text/plain")) {
            System.out.println("This is plain text");
            System.out.println("---------------------------");
            System.out.println((String) p.getContent());
         }
         //check if the content has attachment
         else if (p.isMimeType("multipart/*")) {
            System.out.println("This is a Multipart");
            System.out.println("---------------------------");
            Multipart mp = (Multipart) p.getContent();
            int count = mp.getCount();
            for (int i = 0; i < count; i++)
               writePart(mp.getBodyPart(i));
         }
         //check if the content is a nested message
         else if (p.isMimeType("message/rfc822")) {
            System.out.println("This is a Nested Message");
            System.out.println("---------------------------");
            writePart((Part) p.getContent());
         }
         //check if the content is an inline image
```

```
        else if (p.isMimeType("image/jpeg")) {
           System.out.println("--------> image/jpeg");
           Object o = p.getContent();

           InputStream x = (InputStream) o;
           // Construct the required byte array
           System.out.println("x.length = " + x.available());
           int i = 0;
           byte[] bArray = new byte[x.available()];

           while ((i = (int) ((InputStream) x).available()) > 0) {
              int result = (int) (((InputStream) x).read(bArray));
              if (result == -1)
              break;
           }
           FileOutputStream f2 = new FileOutputStream("/tmp/image.jpg
           f2.write(bArray);
        }
        else if (p.getContentType().contains("image/")) {
           System.out.println("content type" + p.getContentType());
           File f = new File("image" + new Date().getTime() + ".jpg")
           DataOutputStream output = new DataOutputStream(
              new BufferedOutputStream(new FileOutputStream(f)));
              com.sun.mail.util.BASE64DecoderStream test =
                   (com.sun.mail.util.BASE64DecoderStream) p
                    .getContent();
           byte[] buffer = new byte[1024];
           int bytesRead;
           while ((bytesRead = test.read(buffer)) != -1) {
              output.write(buffer, 0, bytesRead);
           }
        }
        else {
           Object o = p.getContent();
           if (o instanceof String) {
              System.out.println("This is a string");
              System.out.println("---------------------------");
              System.out.println((String) o);
           }
           else if (o instanceof InputStream) {
              System.out.println("This is just an input stream");
              System.out.println("---------------------------");
              InputStream is = (InputStream) o;
              is = (InputStream) o;
              int c;
              while ((c = is.read()) != -1)
                 System.out.write(c);
           }
           else {
              System.out.println("This is an unknown type");
              System.out.println("---------------------------");
              System.out.println(o.toString());
           }
```

```
        }

    }
    /*
    * This method would print FROM,TO and SUBJECT of the message
    */
    public static void writeEnvelope(Message m) throws Exception {
       System.out.println("This is the message envelope");
       System.out.println("---------------------------");
       Address[] a;

       // FROM
       if ((a = m.getFrom()) != null) {
          for (int j = 0; j < a.length; j++)
          System.out.println("FROM: " + a[j].toString());
       }

       // TO
       if ((a = m.getRecipients(Message.RecipientType.TO)) != null)
          for (int j = 0; j < a.length; j++)
          System.out.println("TO: " + a[j].toString());
       }

       // SUBJECT
       if (m.getSubject() != null)
          System.out.println("SUBJECT: " + m.getSubject());

    }

 }
```

> 您可以通过取消注释语句上设置调试emailSession.setDebug(true);

# 编译并运行

现在，我们班是准备好了，让我们编译上面的类。我已经保存了类 FetchingEmail.java目录： /home/manisha/JavaMailAPIExercise. 我们需要 javax.mail.jar andactivation.jar 在classpath中。执行下面的命令从命令提示符编译类（两个罐子被放置在 /home/manisha/目录下）：

```
 javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.ja
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

# 验证输出

你应该看到下面的消息命令控制台上：

```
messages.length---3
---------------------------------
This is the message envelope
---------------------------
FROM: XYZ <xyz@gmail.com>
TO: ABC <abc@gmail.com>
SUBJECT: Simple Message
----------------------------
CONTENT-TYPE: multipart/alternative; boundary=047d7b343d6ad3e4ea04e
This is a Multipart
---------------------------

----------------------------
CONTENT-TYPE: text/plain; charset=ISO-8859-1
This is plain text
---------------------------
Hi am a simple message string....

--
Regards
xyz

This is the message envelope
---------------------------
FROM: XYZ <xyz@gmail.com>
TO: ABC <abc@gmail.com>
SUBJECT: Attachement
----------------------------
CONTENT-TYPE: multipart/mixed; boundary=047d7b343d6a99180904e8ec675
This is a Multipart
---------------------------

----------------------------
CONTENT-TYPE: text/plain; charset=ISO-8859-1
This is plain text
---------------------------
Hi I've an attachment.Please check

--
Regards
XYZ
```

```
----------------------------
CONTENT-TYPE: application/octet-stream; name=sample_attachement
This is just an input stream
---------------------------
Submit your Tutorials, White Papers and Articles into our Tutorials

This is the message envelope
---------------------------
FROM: XYZ <xyz@gmail.com>
TO: ABC <abc@gmail.com>
SUBJECT: Inline Image
----------------------------
CONTENT-TYPE: multipart/related; boundary=f46d04182582be803504e8ece
This is a Multipart
---------------------------


----------------------------
CONTENT-TYPE: text/plain; charset=ISO-8859-1
This is plain text
---------------------------
Hi I've an inline image

[image: Inline image 3]

--
Regards
XYZ

----------------------------
CONTENT-TYPE: image/png; name="javamail-mini-logo.png"
content typeimage/png; name="javamail-mini-logo.png"
```

在这里，你可以看到有三封邮件在邮箱中。首先一个简单的邮件消息 "Hi am a simple message string....". 第二个邮件有附件。附件的内容也印如上所示。第三个邮件有一个内嵌图像。

# JavaMail认证/验证 - JavaMail

在前面的章节查询电子邮件和读取电子邮件，我们通过授权凭证（用户名和密码）以及主机，连接到您的邮箱中存储时。相反，我们可以配置属性有主机，并告诉您的自定义的Authenticator实例的会话。这示于下面的例子：

## 创建Java类

我们将修改CheckingMails.java 请先阅读 章查询电子邮件 这章。其内容如下所述：

```
package com.yiibai;

import java.util.Properties;

import javax.mail.Authenticator;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Store;

public class CheckingMails {

   public static void check(String host, String storeType, String u
      String password)
   {
      try {

      // create properties field
      Properties properties = new Properties();

      properties.put("mail.pop3s.host", host);
      properties.put("mail.pop3s.port", "995");
      properties.put("mail.pop3s.starttls.enable", "true");

      // Setup authentication, get session
      Session emailSession = Session.getInstance(properties,
         new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentica
               return new PasswordAuthentication(
                  "manishapatil3may@gmail.com", "manisha123");
            }
         });
      // emailSession.setDebug(true);
```

```
        // create the POP3 store object and connect with the pop serv
        Store store = emailSession.getStore("pop3s");

        store.connect();

        // create the folder object and open it
        Folder emailFolder = store.getFolder("INBOX");
        emailFolder.open(Folder.READ_ONLY);

        // retrieve the messages from the folder in an array and prir
        Message[] messages = emailFolder.getMessages();
        System.out.println("messages.length---" + messages.length);

        for (int i = 0, n = messages.length; i < n; i++) {
           Message message = messages[i];
           System.out.println("---------------------------------");
           System.out.println("Email Number " + (i + 1));
           System.out.println("Subject: " + message.getSubject());
           System.out.println("From: " + message.getFrom()[0]);
           System.out.println("Text: " + message.getContent().toStrir
        }

        // close the store and folder objects
        emailFolder.close(false);
        store.close();

        } catch (NoSuchProviderException e) {
           e.printStackTrace();
        } catch (MessagingException e) {
           e.printStackTrace();
        } catch (Exception e) {
           e.printStackTrace();
        }
     }

     public static void main(String[] args) {

        String host = "pop.gmail.com";// change accordingly
        String mailStoreType = "pop3";
        String username = "abc@gmail.com";// change accordingly
        String password = "*****";// change accordingly

        check(host, mailStoreType, username, password);

     }

  }
```

> 您可以通过取消注释语句上设置调试emailSession.setDebug(true);

# 编译并运行

现在，我们班是准备好了，让我们编译上面的类。我已经保存了类 CheckingMails.java目录： /home/manisha/JavaMailAPIExercise. 我们需要 javax.mail.jar andactivation.jar 在classpath中。执行下面的命令从命令提示符编译类（两个罐子被放置在 /home/manisha/ 目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

# 验证输出

你可以看到一个类似的消息，如下命令控制台上：

```
messages.length---3
---------------------------------
Email Number 1
Subject: Today is a nice day
From: XYZ <xyz@gmail.com>
Text: javax.mail.internet.MimeMultipart@45f676cb
---------------------------------
Email Number 2
Subject: hiiii....
From: XYZ <xyz@gmail.com>
Text: javax.mail.internet.MimeMultipart@37f12d4f
---------------------------------
Email Number 3
Subject: helloo
From: XYZ <xyz@gmail.com>
Text: javax.mail.internet.MimeMultipart@3ad5ba3a
```

# JavaMail 电子邮件答复/回复 - JavaMail

在本章中，我们将看到如何使用JavaMail API来回复电子邮件。接着在下面的程序中的列出基本步骤：

- 获取Session对象与POP和SMTP 服务器的细节属性。我们需要 POP 细节来检索信息和SMPT详细信息发送邮件。
- 创建POP3存储对象，并连接到存储。
- 创建文件夹对象，并在您的邮箱中打开相应的文件夹。
- 检索消息。
- 遍历的消息，如果你想回复键入“Y”或“y”。
- 得到消息的所有信息（收件人，发件人，主题，内容）(To,From,Subject, Content) 。
- 建立应答消息，使用Message.reply()方法。这个方法配置一个新的消息与适当的收件人和主题。该方法接受一个布尔参数，指示是否只回复给发送者 (false) 或回复给所有人（true）。
- 从设置，文本和回复到邮件中，并通过传输对象的实例发送。
- 关闭传输，文件夹和存储对象分别。

> 在这里，我们使用JangoSMPT服务器通过该电子邮件发送到我们的目标电子邮件地址。设置是在环境设置章节解释。

## 创建Java类

创建一个Java类文件ReplyToEmail，是其内容如下：

```
package com.yiibai;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Store;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
```

```
public class ReplyToEmail {
   public static void main(String args[])
   {
      Date date = null;

      Properties properties = new Properties();
      properties.put("mail.store.protocol", "pop3");
      properties.put("mail.pop3s.host", "pop.gmail.com");
      properties.put("mail.pop3s.port", "995");
      properties.put("mail.pop3.starttls.enable", "true");
      properties.put("mail.smtp.auth", "true");
      properties.put("mail.smtp.starttls.enable", "true");
      properties.put("mail.smtp.host", "relay.jangosmtp.net");
      properties.put("mail.smtp.port", "25");
      Session session = Session.getDefaultInstance(properties);

      // session.setDebug(true);
      try
      {
         // Get a Store object and connect to the current host
         Store store = session.getStore("pop3s");
         store.connect("pop.gmail.com", "xyz@gmail.com",
            "*****");//change the user and password accordingly

         Folder folder = store.getFolder("inbox");
         if (!folder.exists()) {
            System.out.println("inbox not found");
               System.exit(0);
         }
         folder.open(Folder.READ_ONLY);

         BufferedReader reader = new BufferedReader(new InputStrear
            System.in));

         Message[] messages = folder.getMessages();
         if (messages.length != 0) {

            for (int i = 0, n = messages.length; i < n; i++) {
               Message message = messages[i];
               date = message.getSentDate();
               // Get all the information from the message
               String from = InternetAddress.toString(message.getFr
               if (from != null) {
                  System.out.println("From: " + from);
               }
               String replyTo = InternetAddress.toString(message
              .getReplyTo());
               if (replyTo != null) {
                  System.out.println("Reply-to: " + replyTo);
               }
               String to = InternetAddress.toString(message
              .getRecipients(Message.RecipientType.TO));
```

```
               if (to != null) {
                  System.out.println("To: " + to);
               }

               String subject = message.getSubject();
               if (subject != null) {
                  System.out.println("Subject: " + subject);
               }
               Date sent = message.getSentDate();
               if (sent != null) {
                  System.out.println("Sent: " + sent);
               }

               System.out.print("Do you want to reply [y/n] : ");
               String ans = reader.readLine();
               if ("Y".equals(ans) || "y".equals(ans)) {

                  Message replyMessage = new MimeMessage(session);
                  replyMessage = (MimeMessage) message.reply(false
                  replyMessage.setFrom(new InternetAddress(to));
                  replyMessage.setText("Thanks");
                  replyMessage.setReplyTo(message.getReplyTo());

                  // Send the message by authenticating the SMTP se
                  // Create a Transport instance and call the sendM
                  Transport t = session.getTransport("smtp");
                  try {
                 //connect to the smpt server using transport instan
              //change the user and password accordingly
                 t.connect("abc", "****");
                 t.sendMessage(replyMessage,
                        replyMessage.getAllRecipients());
                  } finally {
                     t.close();
                  }
                  System.out.println("message replied successfully

                  // close the store and folder objects
                  folder.close(false);
                  store.close();

               } else if ("n".equals(ans)) {
                  break;
               }
            }//end of for loop

         } else {
            System.out.println("There is no msg....");
         }

      } catch (Exception e) {
         e.printStackTrace();
      }
```

```
    }
}
```

> 您可以通过取消注释语句上设置调试 session.setDebug(true);

## 编译并运行

现在，我们班是准备好了，让我们编译上面的类。我已经保存了类 ReplyToEmail.java到目录: /home/manisha/JavaMailAPIExercise. 我们需要 javax.mail.jar andactivation.jar 在 classpath 中。执行下面的命令从命令提示符编译类（两个罐子被放置在 /home/manisha/ 目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

现在，这个类被编译，执行下面的命令来运行：

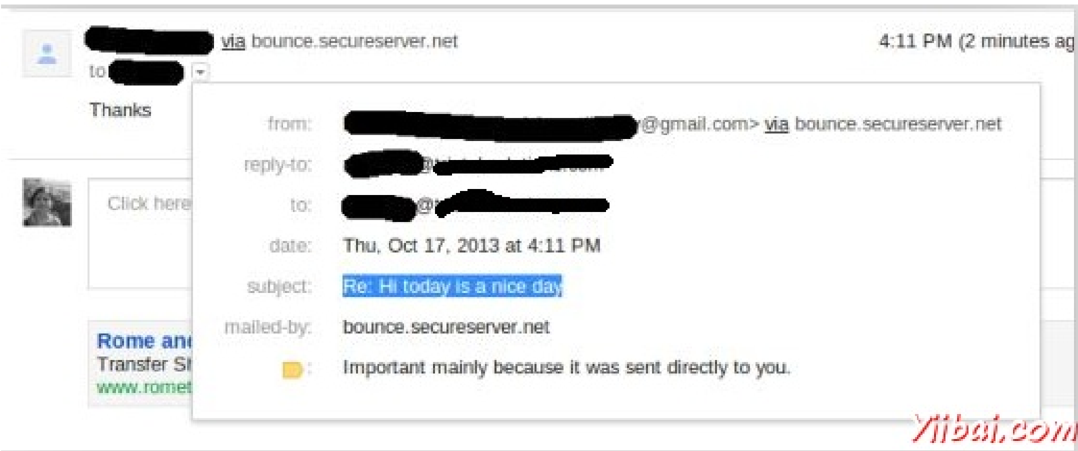
```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

## 验证输出

你应该看到下面的消息命令控制台上：

```
From: ABC <abc@gmail.com>
Reply-to: abc@trioteksolutions.com
To: XYZ <xyz@gmail.com>
Subject: Hi today is a nice day
Sent: Thu Oct 17 15:58:37 IST 2013
Do you want to reply [y/n] : y
message replied successfully ....
```

检查该邮件发送的收件箱。在我们的例子中收到的邮件看起来如下：



via bounce.secureserver.net
4:11 PM (2 minutes ag
to
Thanks
from:
@gmail.com> via bounce.secureserver.net
reply-to:
to:
date:
Thu, Oct 17, 2013 at 4:11 PM
subject:
Re: Hi today is a nice day
mailed-by:
bounce.secureserver.net
Important mainly because it was sent directly to you.
Click here
Yiibai.com

# JavaMail 转发电子邮件 - JavaMail

在本章中，我们将看到如何使用JavaMail API来转发电子邮件。接着在下面的程序的基本步骤是：

- 获取Session对象与POP和SMTP服务器的细节的属性。我们需要的POP细节来检索信息和SMPT详细信息发送邮件。
- 创建POP3存储对象，并连接到存储。
- 创建文件夹对象，并在您的邮箱中打开相应的文件夹。
- 检索消息。
- 遍历的消息，如果你想转发键入“Y”或“y”。
- 得到消息的所有信息（收件人，发件人，主题，内容）。
- 通过与组成消息的各个部分的工作建立转发消息。第一部分将是消息的文本和第二部分将要转发的邮件。结合两成多部分。那么你多部分添加到妥善处理消息并发送它。
- 关闭传输，文件夹和存储对象分别。

> 在这里，我们使用JangoSMPT服务器通过该电子邮件被发送到我们的目标电子邮件地址。设置是在环境设置章节解释。

## 创建Java类

创建一个Java类文件ForwardEmail，是其内容如下：

```
package com.yiibai;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.Properties;

import javax.mail.BodyPart;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Store;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
```

```
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

public class ForwardEmail {

   public static void main(String[] args) {
      Properties properties = new Properties();
      properties.put("mail.store.protocol", "pop3");
      properties.put("mail.pop3s.host", "pop.gmail.com");
      properties.put("mail.pop3s.port", "995");
      properties.put("mail.pop3.starttls.enable", "true");
      properties.put("mail.smtp.auth", "true");
      properties.put("mail.smtp.host", "relay.jangosmtp.net");
      properties.put("mail.smtp.port", "25");
      Session session = Session.getDefaultInstance(properties);
      try {
         // session.setDebug(true);
         // Get a Store object and connect to the current host
         Store store = session.getStore("pop3s");
         store.connect("pop.gmail.com", "xyz@gmail.com",
            "*****");//change the user and password accordingly

         // Create a Folder object and open the folder
         Folder folder = store.getFolder("inbox");
         folder.open(Folder.READ_ONLY);
         BufferedReader reader = new BufferedReader(new InputStrear
            System.in));
         Message[] messages = folder.getMessages();
         if (messages.length != 0) {

         for (int i = 0, n = messages.length; i < n; i++) {
            Message message = messages[i];
            // Get all the information from the message
            String from = InternetAddress.toString(message.getFrom(
            if (from != null) {
               System.out.println("From: " + from);
            }
            String replyTo = InternetAddress.toString(message
               .getReplyTo());
            if (replyTo != null) {
               System.out.println("Reply-to: " + replyTo);
            }
            String to = InternetAddress.toString(message
               .getRecipients(Message.RecipientType.TO));
            if (to != null) {
               System.out.println("To: " + to);
            }

            String subject = message.getSubject();
            if (subject != null) {
               System.out.println("Subject: " + subject);
            }
            Date sent = message.getSentDate();
```

```
            if (sent != null) {
               System.out.println("Sent: " + sent);
            }
            System.out.print("Do you want to reply [y/n] : ");
            String ans = reader.readLine();
            if ("Y".equals(ans) || "y".equals(ans)) {
               Message forward = new MimeMessage(session);
               // Fill in header
               forward.setRecipients(Message.RecipientType.TO,
               InternetAddress.parse(from));
               forward.setSubject("Fwd: " + message.getSubject());
               forward.setFrom(new InternetAddress(to));

               // Create the message part
               MimeBodyPart messageBodyPart = new MimeBodyPart();
               // Create a multipart message
               Multipart multipart = new MimeMultipart();
               // set content
               messageBodyPart.setContent(message, "message/rfc822'
               // Add part to multi part
               multipart.addBodyPart(messageBodyPart);
               // Associate multi-part with message
               forward.setContent(multipart);
               forward.saveChanges();

               // Send the message by authenticating the SMTP serve
               // Create a Transport instance and call the sendMess
               Transport t = session.getTransport("smtp");
               try {
                  //connect to the smpt server using transport inst
          //change the user and password accordingly
                  t.connect("abc", "*****");
                  t.sendMessage(forward, forward.getAllRecipients(
               } finally {
                  t.close();
               }

               System.out.println("message forwarded successfully.

            // close the store and folder objects
            folder.close(false);
            store.close();
            }// end if

         }// end for
   }// end if
   } catch (Exception e) {
      e.printStackTrace();
   }
}

}
```

> 您可以通过取消注释语句上设置调试 session.setDebug(true);

## 编译并运行

现在类准备好了，编译上面的类。我已经保存了类ForwardEmail.java目录：/home/manisha/JavaMailAPIExercise. 我们需要javax.mail.jar 和 activation.jar 在classpath中。执行下面的命令从命令提示符编译类（两个jar 放置在 /home/manisha/ 目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

## 验证输出

你应该看到下面的消息命令控制台上：

```
From: ABC <abc@gmail.com>
Reply-to: abc@trioteksolutions.com
To: XYZ <xyz@gmail.com>
Subject: Hi today is a nice day
Sent: Thu Oct 17 15:58:37 IST 2013
Do you want to reply [y/n] : y
message forwarded successfully....
```

检查该邮件发送的收件箱。在我们的例子中转发的邮件看起来如下：

via jangomail.com 4:37 PM (1 minute ago)

to me

from: @gmail.com> via jangomail.com

to: @gmail.com>

date: Thu, Oct 17, 2013 at 4:37 PM

subject: Fwd: Hi today is a nice day

mailed-by: jangomail.com

Important mainly because of your interaction with messages in the conversation.



Hi today is a nice day Inbox x

via jangomail.com

to me

---------- Forwarded message ----------
From: @gmail.com>
To: @gmail.com>
Cc:
Date: Thu, 17 Oct 2013 15:58:37 +0530
Subject: Hi today is a nice day

Hello to day is a noce day
--
Regards
XYZ

# JavaMail 删除电子邮件 - JavaMail

在本章中，我们将看到如何使用JavaMail API来删除电子邮件。删除信息涉及与该消息相关联的标志工作。有不同的标志为不同的状态，一些系统定义和一些用户定义的。预定义的标志在内部类中定义的标志。标志如下所列：

- Flags.Flag.ANSWERED
- Flags.Flag.DELETED
- Flags.Flag.DRAFT
- Flags.Flag.FLAGGED
- Flags.Flag.RECENT
- Flags.Flag.SEEN
- Flags.Flag.USER

POP协议支持的消息的唯一删除。

其次在删除程序的基本步骤是：

- 获取Session对象与POP和SMTP伺服器的细节的属性。我们需要的POP细节来检索信息和SMPT详细信息发送邮件。
- 创建POP3存储对象，并连接到存储。
- 创建文件夹对象，并在READ_WRITE模式下邮箱打开相应的文件夹。
- 从收件箱文件夹中检索邮件。
- 遍历的消息，如果你想通过Message对象上调用方法setFlag(Flags.Flag.DELETED, true)以删除邮件中键入“Y”或“y”。
- 这些消息标记DELETED 实际上并没有删除，直到我们调用Folder对象上expunge() 方法，或expunge 设置为true，关闭文件夹。
- 关闭存储对象。

## 创建Java类

创建一个Java类文件ForwardEmail，是其内容如下：

```
package com.yiibai;

import java.io.BufferedReader;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.util.Properties;

import javax.mail.Flags;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.Session;
import javax.mail.Store;

public class DeleteEmail {

   public static void delete(String pop3Host, String storeType, Str
      String password)
   {
      try
      {
         // get the session object
         Properties properties = new Properties();
         properties.put("mail.store.protocol", "pop3");
         properties.put("mail.pop3s.host", pop3Host);
         properties.put("mail.pop3s.port", "995");
         properties.put("mail.pop3.starttls.enable", "true");
         Session emailSession = Session.getDefaultInstance(properti
         // emailSession.setDebug(true);

         // create the POP3 store object and connect with the pop s
         Store store = emailSession.getStore("pop3s");

         store.connect(pop3Host, user, password);

         // create the folder object and open it
         Folder emailFolder = store.getFolder("INBOX");
         emailFolder.open(Folder.READ_WRITE);

         BufferedReader reader = new BufferedReader(new InputStrear
            System.in));
         // retrieve the messages from the folder in an array and p
         Message[] messages = emailFolder.getMessages();
         System.out.println("messages.length---" + messages.length)
         for (int i = 0; i < messages.length; i++) {
            Message message = messages[i];
            System.out.println("---------------------------------")
            System.out.println("Email Number " + (i + 1));
            System.out.println("Subject: " + message.getSubject());
            System.out.println("From: " + message.getFrom()[0]);

            String subject = message.getSubject();
            System.out.print("Do you want to delete this message [y
            String ans = reader.readLine();
            if ("Y".equals(ans) || "y".equals(ans)) {
           // set the DELETE flag to true
```

```
            message.setFlag(Flags.Flag.DELETED, true);
            System.out.println("Marked DELETE for message: " + subje
             } else if ("n".equals(ans)) {
            break;
             }
          }
          // expunges the folder to remove messages which are markec
          emailFolder.close(true);
          store.close();

       } catch (NoSuchProviderException e) {
          e.printStackTrace();
       } catch (MessagingException e) {
          e.printStackTrace();
       } catch (IOException io) {
          io.printStackTrace();
       }
    }

    public static void main(String[] args) {

       String host = "pop.gmail.com";// change accordingly
       String mailStoreType = "pop3";
       String username = "abc@gmail.com";// change accordingly
       String password = "*****";// change accordingly

       delete(host, mailStoreType, username, password);

    }

 }
```

> 您可以通过取消注释语句上设置调试 emailSession.setDebug(true);

## 编译并运行

现在，我们的类是准备好了，让我们编译上面的类。我已经保存了类 DeleteEmail.java目录：/home/manisha/JavaMailAPIExercise. 我们需要 javax.mail.jar 和 activation.jar 在 classpath 中。执行下面的命令从命令提示符编译类（两个jar 放置在 /home/manisha/ 目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jai
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

# 验证输出

你应该看到下面的消息命令控制台上：

```
messages.length---1
---------------------------------
Email Number 1
Subject: Testing
From: ABC <abc@gmail.com>
Do you want to delete this message [y/n] ? y
Marked DELETE for message: Testing
```

# JavaMail Gmail SMTP服务器 - JavaMail

在所有前面的章节中，我们使用JangoSMPT服务器来发送电子邮件。在本章中，我们将了解通过Gmail时提供的SMTP伺服器。 Gmail的（等等）提供了使用他们的公共SMTP服务器的免费。

Gmail SMTP服务器的详细信息可以在这里找到。正如你可以在细节里看到的一样，我们可以使用TLS或SSL连接，以通过Gmail SMTP服务器发送邮件。

使用Gmail SMTP服务器发送邮件的过程类似的发送电子邮件的章节中描述说明，除了我们改变主机服务器。作为先决条件，发件人的电子邮件地址应该是一个活跃的Gmail帐户。让我们尝试一个例子。

## 创建Java类

创建一个Java类文件SendEmailUsingGMailSMTP，内容都是如下：

```
package com.yiibai;

import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendEmailUsingGMailSMTP {
   public static void main(String[] args) {
      // Recipient's email ID needs to be mentioned.
      String to = "xyz@gmail.com";//change accordingly

      // Sender's email ID needs to be mentioned
      String from = "abc@gmail.com";//change accordingly
      final String username = "abc";//change accordingly
      final String password = "*****";//change accordingly

      // Assuming you are sending email through relay.jangosmtp.net
      String host = "smtp.gmail.com";

      Properties props = new Properties();
      props.put("mail.smtp.auth", "true");
      props.put("mail.smtp.starttls.enable", "true");
      props.put("mail.smtp.host", host);
      props.put("mail.smtp.port", "587");
```

```
        // Get the Session object.
        Session session = Session.getInstance(props,
        new javax.mail.Authenticator() {
           protected PasswordAuthentication getPasswordAuthentication
              return new PasswordAuthentication(username, password);
           }
        });

        try {
           // Create a default MimeMessage object.
           Message message = new MimeMessage(session);

           // Set From: header field of the header.
           message.setFrom(new InternetAddress(from));

           // Set To: header field of the header.
           message.setRecipients(Message.RecipientType.TO,
           InternetAddress.parse(to));

           // Set Subject: header field
           message.setSubject("Testing Subject");

           // Now set the actual message
           message.setText("Hello, this is sample for to check send "
              + "email using JavaMailAPI ");

           // Send message
           Transport.send(message);

           System.out.println("Sent message successfully....");

        } catch (MessagingException e) {
              throw new RuntimeException(e);
        }
     }
  }
```

主机设置为smtp.gmail.com，端口设置为587。在这里，我们已经启用TLS连接。

## 编译并运行

现在，我们的类是准备好了，让我们编译上面的类。我已经保存了类SendEmailUsingGMailSMTP.java到目录: /home/manisha/JavaMailAPIExercise. 我们需要 javax.mail.jar 和 activation.jar 文件在classpath中。执行下面的命令从命令提示符编译类（jar文件放置在 /home/manisha/目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

## 验证输出

你应该可以看到下面的消息命令控制台上：

```
Sent message successfully....
```

# JavaMail 邮件文件夹管理 - JavaMail

到目前为止，我们已经在前面的章节主要介绍收件箱文件夹的工作。这是大多数邮件位于其中的默认文件夹。有些系统可能会调用它的收件箱和其他一些可能被其他一些名字来称呼它。但是，你总是可以从JavaMail API为使用该名称的收件箱访问它。

JavaMail API为代表文件夹的文件夹抽象类的实例：

```
public abstract class Folder extends Object
```

这个类声明请求命名文件夹从服务器，从文件夹中删除邮件，搜索文件夹中特定消息，列出文件夹中的邮件，等等方法。

## 打开文件夹

我们不能直接创建一个文件夹中 Folder 类唯一的构造函数是受保护的。我们可以得到一个文件夹从：

- Session
- Store
- 另外的Folder

上述所有的类都有一个类似的getFolder()方法类似签名：

```
public abstract Folder getFolder(String name) throws MessagingExcep
```

一些在其中获取Folder 对象帮助的方法有：

| 方法 | 描述 |
|---|---|
| boolean *exists()* | Checks if the folder really exists. Use this method before getting the Folder object. |
| abstract void *open*(int mode) | When you get a *Folder*, its closed. Use this method to open it. _mode _can be Folder.READ_ONLY or Folder.READ_WRITE. |
| abstract boolean *isOpen*() | This method returns *true* if the folder is open, *false* if it’s closed |
| abstract void *close*(boolean expunge) | Closes the folder. If the *expunge* argument is *true*, any deleted messages in the folder are deleted from the actual file on the server. Otherwise, they’re simply marked as*deleted*, but the messages can still be undeleted. |

# 基本文件夹信息

以下是一些它返回有关一个文件夹的基本信息，文件夹类中的方法：

| 方法 | 描述 |
|---|---|
| abstract String *getName*() | Returns the name of the folder, such as "TutorialsPoint Mail" |
| abstract String *getFullName*() | Returns the complete hierarchical name from the root such as “books/Manisha/TutorialsPoint Mail”. |
| URLName *getURLName*() | Return a URLName representing this folder. |
| abstract Folder *getParent*() | Returns the name of the folder that contains this folder i.e the parent folder. E.g "Manisha" from the previous "TutorialsPoint Mail" example. |
| abstract int *getType()* | Returns an int indicating whether the folder can contain messages and/or other folders. |
| int *getMode()* | It returns one of the two named constants Folder.READ_ONLY or Folder.READ_WRITE or -1 when the mode is unknown. |
| Store *getStore*() | Returns the Store object from which this folder was retrieved. |
| abstract char *getSeparator*() | Return the delimiter character that separates this Folder's pathname from the names of immediate subfolders. |

## 管理文件夹

以下是一些有助于管理文件夹的方法：

| 方法 | 描述 |
| --- | --- |
| abstract boolean *create*(int type) | This creates a new folder in this folder's Store. Where *type_would be:Folder.HOLDS_MESSAGES or Folder.HOLDS_FOLDERS. Returns _true* if folder is successfully created else returns *false*. |
| abstract boolean *delete*(boolean recurse) | This deletes the folder only if the folder is closed. Otherwise, it throws an *IllegalStateException*. If *recurse* is*true*, then subfolders are deleted. |
| abstract boolean *renameTo*(Folder f) | This changes the name of this folder. A folder must be closed to be renamed. Otherwise, an IllegalStateException is thrown. |

## 在文件夹管理邮件

以下是一些帮助文件夹管理邮件的方法：

| 方法 | 描述 |
| --- | --- |
| abstract void*appendMessages*(Message[] messages) | As the name implies, the messages in the array are placed at the end of this folder. |
| void *copyMessages*(Message[] messages, Folder destination) | This copies messages from this folder into a specified folder given as an argument. |
| abstract Message[] *expunge*() | To delete a message from a folder, set its Flags.Flag.DELETED flag to true. To physically remove deleted messages from a folder, you have to call this method. |

## 列出文件夹的内容

有四种方法可以列出一个文件夹中包含的文件夹：

| 方法 | 描述 |
|---|---|
| Folder[] *list*() | This returns an array listing the folders that this folder contains. |
| Folder[] *listSubscribed*() | This returns an array listing all the subscribed folders that this folder contains. |
| abstract *Folder*[] list(String pattern) | This is similar to the *list()* method except that it allows you to specify a pattern. The pattern is a string giving the name of the folders that match. |
| Folder[] *listSubscribed*(String pattern) | This is similar to the *listSubscribed()* method except that it allows you to specify a pattern. The pattern is a string giving the name of the folders that match. |

## 检查邮件

| 方法 | 描述 |
|---|---|
| abstract int *getMessageCount*() | This method can be invoked on an open or closed folder. However, in the case of a closed folder, this method may (or may not) return -1 to indicate that the exact number of messages isn’t easily available. |
| abstract boolean *hasNewMessages*() | This returns *true* if new messages have been added to the folder since it was last opened. |
| int *getNewMessageCount*() | It returns the new message count by checking messages in the folder whose RECENT flag is set. |
| int *getUnreadMessageCount*() | This can be invoked on either an open or a closed folder. However, in the case of a closed folder, it may return -1 to indicate that the real answer would be too expensive to obtain. |

## 获取信息的文件夹

Folder类提供了四种方法，用于检索从打开文件夹的邮件：

| 方法 | 描述 |
|---|---|
| abstract Message *getMessage*(int messageNumber) | This returns the nth message in the folder. The first message in the folder is number 1. |
| Message[] *getMessages*() | This returns an array of *Message* objects representing all the messages in this folder. |
| Message[] *getMessages*(int start, int end) | This returns an array of *Message* objects from the folder, beginning with start and finishing with end, inclusive. |
| Message[] *getMessages*(int[] messageNumbers) | This returns an array containing only those messages specifically identified by number in the _messageNumbers_array. |
| void *fetch*(Message[] messages, FetchProfile fp) | Prefetch the items specified in the FetchProfile for the given Messages. The FetchProfile argument specifies which headers in the messages to prefetch. |

## 搜索文件夹

如果服务器支持搜索（许多IMAP服务器做最POP服务器没有），很容易搜索的文件夹，以满足某些条件的邮件。标准编码在搜索关键词的对象。以下是两种搜索方法：

| 方法 | 描述 |
|---|---|
| Message[] *search*(SearchTerm term) | Search this Folder for messages matching the specified search criterion. Returns an array containing the matching messages. Returns an empty array if no matches were found. |
| Message[] *search*(SearchTerm term, Message[] messages) | Search the given array of messages for those that match the specified search criterion. Returns an array containing the matching messages. Returns an empty array if no matches were found. The the specified Message objects must belong to this folder. |

## Flags

当你需要改变标志的文件夹对整个消息集标志的修改是很有用的。以下是在文件夹类提供的方法：

| 方法 | 描述 |
|---|---|
| void *setFlags*(Message[] messages, Flags flag, boolean value) | Sets the specified flags on the messages specified in the array. |
| void *setFlags*(int start, int end, Flags flag, boolean value) | Sets the specified flags on the messages numbered from start through end, both start and end inclusive. |
| void *setFlags*(int[] messageNumbers, Flags flag, boolean value) | Sets the specified flags on the messages whose message numbers are in the array. |
| abstract Flags *getPermanentFlags*() | Returns the flags that this folder supports for all messages. |

# JavaMail 限额管理 - JavaMail

JavaMail配额是限定或固定的号码或邮件的数量在电子邮件存储。JavaMail API为每个邮件服务请求调用计数配额。电子邮件服务可以适用下列配额标准：

- 外发邮件（包括附件）的最大大小。
- 邮件信息，包括附件的最大大小。
- 消息的最大大小时，管理员是一个收件人

对于配额管理的JavaMail有以下类别：

| Class | 描述 |
| --- | --- |
| public class Quota | This class represents a set of quotas for a given quota root. Each quota root has a set of resources, represented by the Quota.Resource class. Each resource has a name (for example, "STORAGE"), a current usage, and a usage limit. This has only one method *setResourceLimit(String name, long limit)*. |
| public static class Quota.Resource | Represents an individual resource in a quota root. |
| public interface QuotaAwareStore | An interface implemented by Stores that support quotas. The *getQuota* and *setQuota* methods support the quota model defined by the IMAP QUOTA extension.*GmailSSLStore, GmailStore, IMAPSSLStore, IMAPStore* are the known implementing classes of this interface. |

让我们来看看和例子在下面的章节会检查邮件存储名称，并限制其使用。

## 创建Java类

创建一个Java类文件QuotaExample，是其内容如下：

```
package com.yiibai;

import java.util.Properties;

import javax.mail.Quota;
import javax.mail.Session;
import javax.mail.Store;

import com.sun.mail.imap.IMAPStore;

public class QuotaExample
{
   public static void main(String[] args)
   {
      try
      {
         Properties properties = new Properties();
         properties.put("mail.store.protocol", "imaps");
         properties.put("mail.imaps.port", "993");
         properties.put("mail.imaps.starttls.enable", "true");
         Session emailSession = Session.getDefaultInstance(propert
         // emailSession.setDebug(true);

         // create the IMAP3 store object and connect with the pop
         Store store = emailSession.getStore("imaps");

         //change the user and password accordingly
         store.connect("imap.gmail.com", "abc@gmail.com", "*****")
         IMAPStore imapStore = (IMAPStore) store;
         System.out.println("imapStore ---" + imapStore);

         //get quota
         Quota[] quotas = imapStore.getQuota("INBOX");
         //Iterate through the Quotas
         for (Quota quota : quotas) {
            System.out.println(String.format("quotaRoot:'%s'",
               quota.quotaRoot));
            //Iterate through the Quota Resource
            for (Quota.Resource resource : quota.resources) {
               System.out.println(String.format(
                  "name:'%s', limit:'%s', usage:'%s'", resource.nar
                  resource.limit, resource.usage));
            }
         }
      } catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

这里是连接通过IMAP（imap.gmail.com）服务器的Gmail服务，为IMAPStore实现QuotaAwareStore。一旦你获得了存储对象，获取配额阵列和遍历并打印相关信息。

## 编译并运行

现在，我们的类是准备好了，让我们编译上面的类。我已经保存了类QuotaExample.java到目录：/home/manisha/JavaMailAPIExercise. 我们需要javax.mail.jar 和 activation.jar在classpath中。执行下面的命令从命令提示符编译类（两个jar被放置在/home/manisha/ 目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

## 验证输出

您应该看到类似的消息在命令控制台上：

```
imapStore ---imaps://abc%40gmail.com@imap.gmail.com
quotaRoot:''
name:'STORAGE', limit:'15728640', usage:'513'
```

# JavaMail 退回邮件 - JavaMail

消息可以被退回的几个原因。这个问题是在rfc1211深度讨论。只有服务器可确定特定的邮箱或用户名的不存在。当服务器检测到错误，它会返回一个消息，指出失败的原邮件的发件人的原因。

有覆盖发送状态通知许多互联网标准，但大量的服务器不使用特殊技术来返还这些失败消息支持这些新的标准代替。因此，它变得非常难以关联与导致问题的原始邮件的退回邮件。

JavaMail 包括用于解析传递状态通知的支持。有许多技术和试探法来处理这个问题。其中一个是可变的信封返回路径的技术。可以设置的返回路径中的包封器，如下面的例子。这是在弹跳邮件被发送到的地址。您可能需要将其设置为一个通用的地址，而不同：头，这样你就可以处理远程退回。这可以通过设置mail.smtp.from属性在JavaMail中。

## 创建Java类

创建一个Java类文件SendEmail，是其内容如下：

```
import java.util.Properties;

import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendEmail {
   public static void main(String[] args) throws Exception {
      String smtpServer = "smtp.gmail.com";
      int port = 587;
      final String userid = "youraddress";//change accordingly
      final String password = "*****";//change accordingly
      String contentType = "text/html";
      String subject = "test: bounce an email to a different addres
            "from the sender";
      String from = "youraddress@gmail.com";
      String to = "bouncer@fauxmail.com";//some invalid address
      String bounceAddr = "toaddress@gmail.com";//change accordingl
      String body = "Test: get message to bounce to a separate emai

      Properties props = new Properties();

      props.put("mail.smtp.auth", "true");
      props.put("mail.smtp.starttls.enable", "true");
```

```
        props.put("mail.smtp.host", smtpServer);
        props.put("mail.smtp.port", "587");
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.from", bounceAddr);

        Session mailSession = Session.getInstance(props,
           new javax.mail.Authenticator() {
              protected PasswordAuthentication getPasswordAuthenticat
                 return new PasswordAuthentication(userid, password);
              }
           });

        MimeMessage message = new MimeMessage(mailSession);
        message.addFrom(InternetAddress.parse(from));
        message.setRecipients(Message.RecipientType.TO, to);
        message.setSubject(subject);
        message.setContent(body, contentType);

        Transport transport = mailSession.getTransport();
        try {
           System.out.println("Sending ....");
           transport.connect(smtpServer, port, userid, password);
           transport.sendMessage(message,
              message.getRecipients(Message.RecipientType.TO));
           System.out.println("Sending done ...");
        } catch (Exception e) {
           System.err.println("Error Sending: ");
           e.printStackTrace();

        }
        transport.close();
     }// end function main()
  }
```

在这里，我们可以看到，该属性mail.smtp.from设距离来源地址不同。

# 编译并运行

现在，我们的类是准备好了，让我们编译上面的类。我已经保存了类SendEmail.java到目录：/home/manisha/JavaMailAPIExercise. 我们需要javax.mail.jar 和 activation.jar 在classpath中。执行下面的命令从命令提示符编译类（两个jar文件 被放置在/home/manisha/目录下）：

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

现在，这个类被编译，执行下面的命令来运行：

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar
```

## 验证输出

应该看到下面的消息命令控制台上：

```
Sending ....
Sending done ...
```

# JavaMail SMTP服务器 - JavaMail

SMTP是一个缩写为简单邮件传输协议。它是跨越互联网协议（IP）网络电子邮件（电子邮件）传输的Internet标准。 SMTP使用TCP端口25。受SSL保护的SMTP连接通过缩写SMTPS称，虽然SMTPS不是一个协议在自己的权利。

JavaMail API 有包 com.sun.mail.smtp 它作为SMTP协议提供访问SMTP服务器。下表列出了包含在这个包中的类：

| 类 | 描述 |
| --- | --- |
| SMTPMessage | This class is a specialization of the MimeMessage class that allows you to specify various SMTP options and parameters that will be used when this message is sent over SMTP. |
| SMTPSSLTransport | This class implements the Transport abstract class using SMTP over SSL for message submission and transport. |
| SMTPTransport | This class implements the Transport abstract class using SMTP for message submission and transport. |

下表列出抛出的异常：

| 异常 | 描述 |
| --- | --- |
| SMTPAddressFailedException | This exception is thrown when the message cannot be sent. |
| SMTPAddressSucceededException | This exception is chained off a SendFailedException when the *mail.smtp.reportsuccess* property is true. |
| SMTPSenderFailedException | This exception is thrown when the message cannot be sent. |
| SMTPSendFailedException | This exception is thrown when the message cannot be sent.The exception includes the sender's address, which the mail server rejected. |

com.sun.mail.smtp提供使用SMTP身份验证选择性。使用SMTP验证，您需要设置mail.smtp.auth属性或提供SMTP传输与连接到SMTP服务器的用户名和密码。您可以使用以下方法之一进行：

```
Transport.send(message);
```

- 创建您的邮件会话时提供一个Authenticator对象，并提供身份验证器回调过程中的用户名和密码信息。mail.smtp.user属性可以设置，以提供一个默认的用户名回调，但密码仍然需要显式提供。这种方法允许您使用静态传输send方法来发送消息。例如：
- 调用传输用户名和密码参数显式连接方法。例如：

```
Transport tr = session.getTransport("smtp");
tr.connect(smtphost, username, password);
msg.saveChanges();
tr.sendMessage(msg, msg.getAllRecipients());
tr.close();
```

SMTP协议提供程序支持以下属性，这些属性可以在JavaMail会话对象进行设置。该属性始终设置为字符串。例如：

```
props.put("mail.smtp.port", "587");
```

在这里，类型列描述字符串是如何解释的。

| 名称 | 类型 | 描 |
|---|---|---|
| mail.smtp.user | String | Default user name for |
| mail.smtp.host | String | The SMTP server to c |
| mail.smtp.port | int | The SMTP server por connect() method doe one. Defaults to 25. |
| mail.smtp.connectiontimeout | int | Socket connection tim milliseconds. Default i |
| mail.smtp.timeout | int | Socket I/O timeout va Default is infinite time |
| mail.smtp.from | String | Email address to use command. This sets t address. Defaults to n InternetAddress.getLo |
| mail.smtp.localhost | String | Local host name used EHLO command. Defa InetAddress.getLocalI Should not normally n JDK and your name s |

| | | |
|---|---|---|
| | | properly. |
| mail.smtp.localaddress | String | Local address (host n creating the SMTP so address picked by the not normally need to b |
| mail.smtp.localport | int | Local port number to b the SMTP socket. Def number picked by the |
| mail.smtp.ehlo | boolean | If false, do not attemp EHLO command. Defa |
| mail.smtp.auth | boolean | If true, attempt to auth the AUTH command. |
| mail.smtp.auth.mechanisms | String | If set, lists the authent consider. Only mecha server and supported implementation will be "LOGIN PLAIN DIGES includes all the authen supported by the curre |
| mail.smtp.auth.login.disable | boolean | If true, prevents use o command. Default is f |
| mail.smtp.auth.plain.disable | boolean | If true, prevents use o command. Default is f |
| mail.smtp.auth.digest-md5.disable | boolean | If true, prevents use o MD5 command. Defau |
| mail.smtp.auth.ntlm.disable | boolean | If true, prevents use o command. Default is f |
| mail.smtp.auth.ntlm.domain | String | The NTLM authentica |
| mail.smtp.auth.ntlm.flags | int | NTLM protocol-specifi |
| mail.smtp.submitter | String | The submitter to use i MAIL FROM comman mail relay to pass alon original submitter of th |
| mail.smtp.dsn.notify | String | The NOTIFY option to Either NEVER, or son SUCCESS, FAILURE (separated by comma |
| mail.smtp.dsn.ret | String | The RET option to the FULL or HDRS. |

| | | |
|---|---|---|
| mail.smtp.sendpartial | boolean | If set to true, and a m and some invalid addr message anyway, rep with a SendFailedExc (the default), the mess of the recipients if the address. |
| mail.smtp.sasl.enable | boolean | If set to true, attempt javax.security.sasl pa authentication mecha to false. |
| mail.smtp.sasl.mechanisms | String | A space or comma se mechanism names to |
| mail.smtp.sasl.authorizationid | String | The authorization ID t authentication. If not s ID (user name) is use |
| mail.smtp.sasl.realm | String | The realm to use with authentication. |
| mail.smtp.quitwait | boolean | If set to false, the QUI the connection is imm true (the default), cau for the response to th |
| mail.smtp.reportsuccess | boolean | If set to true, causes t an SMTPAddressSuc each address that is s |
| mail.smtp.socketFactory | SocketFactory | If set to a class that in javax.net.SocketFacto will be used to create |
| mail.smtp.socketFactory.class | String | If set, specifies the na implements the javax. interface. This class w SMTP sockets. |
| mail.smtp.socketFactory.fallback | boolean | If set to true, failure to the specified socket fa the socket to be creat java.net.Socket class. |
| mail.smtp.socketFactory.port | int | Specifies the port to c the specified socket fa default port will be use |
| mail.smtp.ssl.enable | boolean | If set to true, use SSL SSL port by default. D |

| | | |
|---|---|---|
| mail.smtp.ssl.enable | boolean | "smtp" protocol and tr protocol. |
| mail.smtp.ssl.checkserveridentity | boolean | If set to true, checks t specified by RFC 259 |
| mail.smtp.ssl.trust | String | If set, and a socket fa specified, enables use MailSSLSocketFactor are trusted. If set to a list of hosts, those ho Otherwise, trust depe the server presents. |
| mail.smtp.ssl.socketFactory | SSLSocketFactory | If set to a class that e javax.net.ssl.SSLSoc class will be used to c sockets. |
| mail.smtp.ssl.socketFactory.class | String | If set, specifies the na extends the javax.net. class. This class will b SSL sockets. |
| mail.smtp.ssl.socketFactory.port | int | Specifies the port to c the specified socket f default port will be use |
| mail.smtp.ssl.protocols | string | Specifies the SSL pro enabled for SSL conn value is a whitespace acceptable to the javax.net.ssl.SSLSoc method. |
| mail.smtp.starttls.enable | boolean | If true, enables the us command (if supporte switch the connection connection before iss commands. Defaults t |
| mail.smtp.starttls.required | boolean | If true, requires the us command. If the serve STARTTLS command the connect method w |
| mail.smtp.socks.host | string | Specifies the host na server that will be use mail server. |
| | | Specifies the port nun |

| | | |
|---|---|---|
| mail.smtp.socks.port | string | used if the proxy serv standard port number |
| mail.smtp.mailextension | String | Extension string to ap command. |
| mail.smtp.userset | boolean | If set to true, use the I instead of the NOOP isConnected method. sendmail will respond NOOP commands; us sendmail issue. Defau |

一般情况下，应用程序应该不需要直接使用的类在这个包。相反，他们应该使用javax.mail的包定义的API（和子包）。例如说，应用程序不应该直接构造SMTPTransport的实例。相反，他们应该使用Session方法getTransport获得一个合适的Transport对象。

示例使用的SMTP服务器是体现在 发送邮件。

# JavaMail IMAP服务器 - JavaMail

IMAP是缩写为Internet邮件访问协议。这是一个应用层互联网协议，它允许电子邮件客户端来访问邮件的远程服务器上的邮件。 IMAP服务器通常监听众所周知的端口143。 IMAP通过SSL（IMAPS）被分配为端口号993。

IMAP支持两种操作在线和离线模式。使用IMAP电子邮件客户端一般在服务器上保留邮件，直到用户显式删除它们。

包装com.sun.mail.imap是IMAP协议提供JavaMail API，它提供了访问一个IMAP邮件存储。下表列出了此提供程序的接口和类：

| 类/接口 | 描述 |
| --- | --- |
| IMAPFolder.ProtocolCommand | This a simple *interface* for user-defined IMAP protocol commands. |
| ACL | This is a class. An access control list entry for a particular authentication identifier (user or group). |
| IMAPFolder | This class implements an IMAP folder. |
| IMAPFolder.FetchProfileItem | This a class for fetching headers. |
| IMAPMessage | This class implements an ReadableMime object. |
| IMAPMessage.FetchProfileCondition | This class implements the test to be done on each message in the folder. |
| IMAPSSLStore | This class provides access to an IMAP message store over SSL. |
| IMAPStore | This class provides access to an IMAP message store. |
| Rights | This class represents the set of rights for an authentication identifier (for instance, a user or a group). |
| Rights.Right | This inner class represents an individual right. |
| SortTerm | A particular sort criteria, as defined by RFC 5256. |

有几点要注意此提供以上：

- 提供同时支持IMAP4和的IMAP4rev1协议。

- 关连IMAPStore维护IMAP协议的对象池在与IMAP服务器进行通信的使用。为文件夹被打开，需要有新的IMAP协议的对象，如果没有可用，IMAPStore将为他们提供从连接池中，或者创建他们。当一个文件夹被关闭，其IMAP协议的对象被返回到连接池，如果池有创建过的对象。
- 该连接IMAPStore对象可能会或可能不会维持一个单独的IMAP协议的对象，它提供了存储专用连接到IMAP服务器。

IMAP协议提供程序支持以下属性，这些属性可以在JavaMail会话对象进行设置。该属性始终设置为字符串;类型列描述字符串是如何解释的。

| 名称 | 类型 | |
|---|---|---|
| mail.imap.user | String | Default user name fo |
| mail.imap.host | String | The IMAP server to |
| mail.imap.port | int | The IMAP server po connect() method do one. Defaults to 143 |
| mail.imap.partialfetch | boolean | Controls whether the capability should be |
| mail.imap.fetchsize | int | Partial fetch size in b |
| mail.imap.ignorebodystructuresize | boolean | The IMAP BODYST includes the exact s Normally, this size is much data to fetch fo Defaults to false. |
| mail.imap.connectiontimeout | int | Socket connection ti milliseconds. Defaul |
| mail.imap.timeout | int | Socket I/O timeout v Default is infinite tim |
| mail.imap.statuscachetimeout | int | Timeout value in mil STATUS command r (1 second). Zero dis |
| mail.imap.appendbuffersize | int | Maximum size of a m memory when appe |
| mail.imap.connectionpoolsize | int | Maximum number o the connection pool. |
| mail.imap.connectionpooltimeout | int | Timeout value in mil pool connections. De seconds). |
| | | Flag to indicate whe |

| | | |
|---|---|---|
| mail.imap.separatestoreconnection | boolean | store connection for<br>is false. |
| mail.imap.auth.login.disable | boolean | If true, prevents use<br>AUTHENTICATE LC<br>using the plain LOG<br>false. |
| mail.imap.auth.plain.disable | boolean | If true, prevents use<br>PLAIN command. D |
| mail.imap.auth.ntlm.disable | boolean | If true, prevents use<br>NTLM command. De |
| mail.imap.proxyauth.user | String | If the server support<br>extension, this prope<br>the user to act as. A<br>using the administra<br>authentication, the II<br>the PROXYAUTH co<br>name specified in th |
| mail.imap.localaddress | String | Local address (host<br>creating the IMAP so<br>address picked by th |
| mail.imap.localport | int | Local port number to<br>the IMAP socket. De<br>picked by the Socke |
| mail.imap.sasl.enable | boolean | If set to true, attemp<br>javax.security.sasl p<br>authentication mech<br>to false. |
| mail.imap.sasl.mechanisms | String | A space or comma s<br>mechanism names t |
| mail.imap.sasl.authorizationid | String | The authorization ID<br>authentication. If not<br>ID (user name) is us |
| mail.imap.sasl.realm | String | The realm to use wit<br>mechanisms that rec<br>DIGEST-MD5. |
| mail.imap.auth.ntlm.domain | String | The NTLM authentic |
| mail.imap.auth.ntlm.flags | int | NTLM protocol-spec |
| mail.imap.socketFactory | SocketFactory | If set to a class that<br>javax.net.SocketFac<br>will be used to creat |

| | | |
|---|---|---|
| mail.imap.socketFactory.class | String | If set, specifies the r implements the java interface. This class IMAP sockets. |
| mail.imap.socketFactory.fallback | boolean | If set to true, failure the specified socket the socket to be crea java.net.Socket clas |
| mail.imap.socketFactory.port | int | Specifies the port to the specified socket used when not set. |
| mail.imap.ssl.enable | boolean | If set to true, use SS SSL port by default. "imap" protocol and protocol. |
| mail.imap.ssl.checkserveridentity | boolean | If set to true, check t specified by RFC 25 |
| mail.imap.ssl.trust | String | If set, and a socket f specified, enables u MailSSLSocketFact are trusted. If set to list of hosts, those h trusted.Otherwise, tr certificate the server |
| mail.imap.ssl.socketFactory | SSLSocketFactory | If set to a class that javax.net.ssl.SSLSo class will be used to sockets. |
| mail.imap.ssl.socketFactory.class | String | If set, specifies the r extends the javax.ne class. This class will SSL sockets. |
| mail.imap.ssl.socketFactory.port | int | Specifies the port to the specified socket default port will be u |
| mail.imap.ssl.protocols | string | Specifies the SSL pr enabled for SSL cor value is a whitespac acceptable to the javax.net.ssl.SSLSo method. |
| | | |

| | | |
|---|---|---|
| mail.imap.starttls.enable | boolean | If true, enables the u command (if suppor switch the connectic connection before is commands. Default |
| mail.imap.starttls.required | boolean | If true, requires the u command. If the ser STARTTLS commar the connect method |
| mail.imap.socks.host | string | Specifies the host na server that will be us mail server. |
| mail.imap.socks.port | string | Specifies the port nu proxy server. This sh used if the proxy ser standard port numbe |
| mail.imap.minidletime | int | This property sets th not set, the default is |
| mail.imap.enableimapevents | boolean | Enable special IMAF delivered to the Stor true, unsolicited resp the Store's idle meth ConnectionEvents w IMAPStore.RESPON message will be the string. By default, th |
| mail.imap.folder.class | String | Class name of a sub com.sun.mail.imap.I subclass can be use additional IMAP com must have public co public MyIMAPFolde separator, IMAPStor isNamespace) and p MyIMAPFolder(ListI |

一般情况下，应用程序应该不需要直接使用的类在这个包。相反，他们应该使用javax.mail包定义的API（和子包）。应用程序不应该兴建IMAPStore或IMAPFolder的直接实例。相反，他们应该使用Session方法getStore获得适当的存储对象，并从获取文件夹对象。

示例使用IMAP服务器是体现在 限额管理.

# JavaMail POP3服务器 - JavaMail

邮局协议（POP）是使用本地电子邮件客户端可以检索电子邮件从远程服务器通过TCP/ IP连接的应用层网络标准协议。 POP支持访问远程邮箱简单的下载和 - 删除要求。一个POP3服务器监听熟知端口110。

软件包com.sun.mail.pop3是POP3协议提供的JavaMail API，它提供了访问的POP3邮件存储。下表列出了类此包中：

| 名称 | 描述 |
| --- | --- |
| POP3Folder | A POP3 Folder (can only be "INBOX"). |
| POP3Message | A POP3 Message. |
| POP3SSLStore | A POP3 Message Store using SSL. |
| POP3Store | A POP3 Message Store. |

有几点要注意此提供以上：

- POP3提供程序只支持一个名为INBOX一个文件夹。由于POP3协议的限制，许多像事件通知，文件夹管理，标志管理等JavaMail的API的功能是不允许的。
- POP3提供通过的JavaMail API的使用协议名称POP3或以下形式的URL访问pop3://user:password@host:port/INBOX".
- POP3支持没有永久的标志。例如，Flags.Flag.RECENT标志将永远不会被用于POP3邮件设置。它是由应用程序，以确定哪些是在一个POP3邮箱的新邮件。
- POP3不支持Folder.expunge()方法。删除并抹去消息，请将Flags.Flag.DELETED标志上的信息，并使用Folder.close（true）方法关闭文件夹。
- POP3不提供接收日期，所以getReceivedDate方法将返回null。
- 当被访问的POP3邮件的标题，在POP3提供程序使用top命令来获取所有标题，然后将其缓存。
- 当访问一个POP3邮件的内容时，POP3提供程序使用RETR命令来获取整个消息。
- POP3Message.invalidate方法可用于缓存的无效数据，但不关闭文件夹。

POP3协议提供程序支持以下属性，这些属性可以在JavaMail会话对象进行设置。该属性始终设置为字符串;类型列描述字符串是如何解释的。

| 名称 | 类型 | 描 |
| --- | --- | --- |

| mail.pop3.user | String | Default user name for |
|---|---|---|
| mail.pop3.host | String | The POP3 server to c |
| mail.pop3.port | int | The POP3 server por connect() method doe one. Defaults to 110. |
| mail.pop3.connectiontimeout | int | Socket connection tim milliseconds. Default i |
| mail.pop3.timeout | int | Socket I/O timeout va Default is infinite time |
| mail.pop3.rsetbeforequit | boolean | Send a POP3 RSET c the folder, before sen command. Default is f |
| mail.pop3.message.class | String | Class name of a subc com.sun.mail.pop3.P( subclass can be used non-standard Content subclass must have a the form MyPOP3Mes msgno) throws Messa |
| mail.pop3.localaddress | String | Local address (host n creating the POP3 so address picked by the |
| mail.pop3.localport | int | Local port number to I the POP3 socket. Def picked by the Socket |
| mail.pop3.apop.enable | boolean | If set to true, use APC USER/PASS to login t the POP3 server supp sends a digest of the the clear text passwor |
| mail.pop3.socketFactory | SocketFactory | If set to a class that in javax.net.SocketFacto will be used to create |
| mail.pop3.socketFactory.class | String | If set, specifies the na implements the javax. interface. This class w POP3 sockets. |
| mail.pop3.socketFactory.fallback | boolean | If set to true, failure to the specified socket fa the socket to be creat |

| | | |
|---|---|---|
| | | java.net.Socket class. |
| mail.pop3.socketFactory.port | int | Specifies the port to c the specified socket fa default port will be us |
| mail.pop3.ssl.enable | boolean | If set to true, use SSL SSL port by default. D "pop3" protocol and tr protocol. |
| mail.pop3.ssl.checkserveridentity | boolean | If set to true, check th specified by RFC 259 |
| mail.pop3.ssl.trust | String | If set, and a socket fa specified, enables use MailSSLSocketFactor are trusted. If set to a list of hosts, those ho Otherwise, trust depe the server presents. |
| mail.pop3.ssl.socketFactory | SSLSocketFactory | If set to a class that e javax.net.ssl.SSLSocl class will be used to c sockets. |
| mail.pop3.ssl.socketFactory.class | String | If set, specifies the na extends the javax.net class. This class will b SSL sockets. |
| mail.pop3.ssl.socketFactory.port | int | Specifies the port to c the specified socket fa default port will be us |
| mail.pop3.ssl.protocols | string | Specifies the SSL pro enabled for SSL conn value is a whitespace acceptable to the javax.net.ssl.SSLSocl method. |
| mail.pop3.starttls.enable | boolean | If true, enables the us command (if supporte switch the connection connection before iss commands. Defaults t |
| mail.pop3.starttls.required | boolean | If true, requires the us command. If the serve STLS command, or th |

| | | |
|---|---|---|
| | | connect method will fa |
| mail.pop3.socks.host | string | Specifies the host nar server that will be use mail server. |
| mail.pop3.socks.port | string | Specifies the port nun proxy server. |
| mail.pop3.disabletop | boolean | If set to true, the POP not be used to fetch n Defaults to false. |
| mail.pop3.forgettopheaders | boolean | If set to true, the head been retrieved using t command will be forg headers retrieved as p command. Defaults to |
| mail.pop3.filecache.enable | boolean | If set to true, the POP message data in a ter in memory. Messages cache when accessin Message headers are memory (on demand) removed when the fol JVM terminates. Defa |
| mail.pop3.filecache.dir | String | If the file cache is ena be used to override th by the JDK for tempor |
| mail.pop3.cachewriteto | boolean | Controls the behavior on a POP3 message and the message con cached, and ignoreLis cached before being w message is streamed stream without being false. |
| mail.pop3.keepmessagecontent | boolean | If this property is set t to the cached content the memory from bein is closed or the cache invalidated (using the Defaults to false. |

一般情况下，应用程序不应在此包中直接使用的类。相反，他们应该使用javax.mail的包定义的API（和子包）。应用程序不应该构建POP3Store或POP3Folder直接的实例。相反，他们应该使用Session方法getStore获得适当的存

储对象，并从获取文件夹对象。

示例使用的POP3服务器是体现在 检查电子邮件.

# JDBC教程

JDBC API是一个Java API可以访问任何类型的表格格式数据，存储在一个关系数据库尤其是数据。 JDBC的工作原理与Java在各种平台一样，如Windows，Mac OS和各种版本的UNIX。

## 读者

本教程是为Java程序员设计的，需要了解JDBC框架的细节以及它的架构和实际使用情况。

## 必备条件

在继续本教程，应该对Java编程语言的一个很好的理解。因为要处理RDBMS，所以它需要对SQL和数据库的概念有很好的理解。

JDBC代表Java数据库连接，这对Java编程语言和广泛的数据库之间独立于数据库的连接标准的Java API。

JDBC库包含的API为每个通常与数据库的使用相关联的任务：

- 使得连接到数据库
- 创建SQL或MySQL语句
- 执行SQL或MySQL的查询数据库
- 查看和修改结果记录

从根本上说，JDBC是一种规范，它提供的接口，一套完整的，允许便携式访问底层数据库。可以用Java来写不同类型的可执行文件，如：

- Java应用程序
- Java Applets
- Java Servlets
- Java ServerPages (JSP)
- Enterprise JavaBeans (EJBs)

所有这些不同的可执行文件就可以使用JDBC驱动程序来访问数据库，并把存储的数据的优势。

JDBC提供了相同的功能，ODBC，允许Java程序包含与数据库无关的代码。

# 先决条件：

以前正如期进行本教程，需要具备以下两个主题内容很好的了解：

- 核心Java编程
- SQL或MySQL数据库

# JDBC架构：

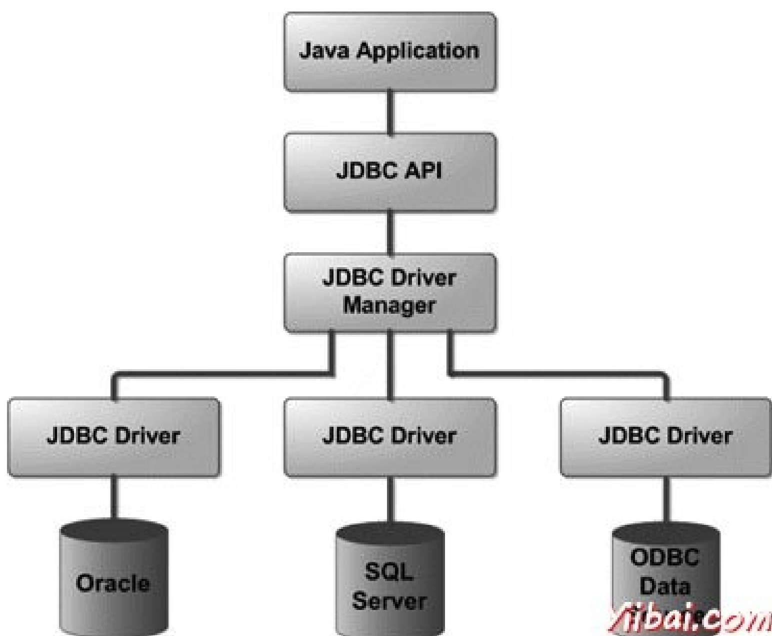
JDBC API支持两层和三层处理模型进行数据库访问，但在一般的JDBC体系结构由两层组成：

- JDBC API: 提供了应用程序对JDBC的管理连接。
- JDBC Driver API: 支持JDBC管理到驱动器连接。

JDBC API的使用驱动程序管理器和数据库特定的驱动程序提供透明的连接到异构数据库。

JDBC驱动程序管理器可确保正确的驱动程序来访问每个数据源。该驱动程序管理器能够支持连接到多个异构数据库的多个并发的驱动程序。

以下是结构图，它显示了驱动程序管理器方面的JDBC驱动程序和Java应用程序的位置：




# 常见的JDBC组件：

JDBC API提供了以下接口和类：

- DriverManager: 这个类管理数据库驱动程序的列表。内容是否符合从Java应用程序使用的通信子协议正确的数据库驱动程序的连接请求。识别JDBC在一定子协议的第一个驱动器将被用来建立数据库连接。
- Driver: 此接口处理与数据库服务器通信。很少直接与驱动程序对象。相反，使用DriverManager中的对象，它管理此类型的对象。它也抽象与驱动程序对象工作相关的详细信息
- Connection : 此接口与接触数据库的所有方法。连接对象表示通信上下文，即，与数据库中的所有的通信是通过唯一的连接对象。
- Statement : 可以使用这个接口创建的对象的SQL语句提交到数据库。一些派生的接口接受除执行存储过程的参数。
- ResultSet: 这些对象保存从数据库后，执行使用Statement对象的SQL查询中检索数据。它作为一个迭代器，让您可以通过移动它的数据。
- SQLException: 这个类处理发生在一个数据库应用程序的任何错误。

# JDBC 4.0软件包

对JDBC4.0，java.sql和javax.sql是主要的包。这是最新版本的JDBC在写这篇教程的时候。它提供的主要类与数据源进行交互。

在这些包中的新功能包括改变在以下几个方面：

- 自动数据库驱动程序加载
- 异常处理的改进
- 增强的BLOB/ CLOB功能
- 连接和语句界面增强
- 国家字符集支持
- SQL ROWID访问
- SQL 2003 XML数据类型支持
- 注释

# JDBC4简介，JDBC是什么？ - JDBC教程

## JDBC是什么?

JDBC代表Java数据库连接，这对Java编程语言和广泛的数据库之间独立于数据库的连接标准的Java API。

JDBC库包含的API为每个通常与数据库的使用相关联的任务：

- 使得连接到数据库
- 创建SQL或MySQL语句
- 执行SQL或MySQL的查询数据库
- 查看和修改结果记录

从根本上说，JDBC是一种规范，它提供的接口，一套完整的，允许便携式访问底层数据库。可以用Java来写不同类型的可执行文件，如：

- Java应用程序
- Java Applets
- Java Servlets
- Java ServerPages (JSP)
- Enterprise JavaBeans (EJBs)

所有这些不同的可执行文件就可以使用JDBC驱动程序来访问数据库，并把存储的数据的优势。

JDBC提供了相同的功能，ODBC，允许Java程序包含与数据库无关的代码。

## 先决条件：

以前正如期进行本教程，需要具备以下两个主题内容很好的了解：

- 核心Java编程
- SQL或MySQL数据库

## JDBC架构：

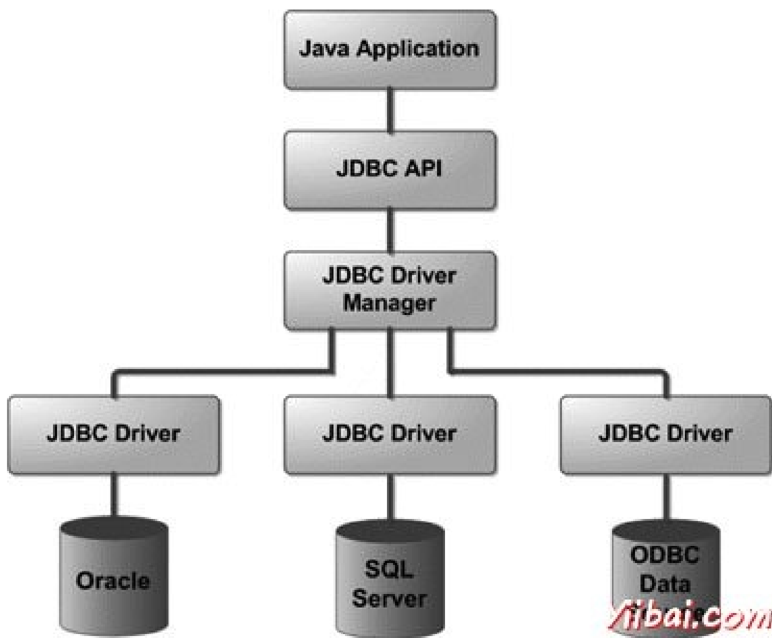
JDBC API支持两层和三层处理模型进行数据库访问，但在一般的JDBC体系结构由两层组成：

- JDBC API: 提供了应用程序对JDBC的管理连接。
- JDBC Driver API: 支持JDBC管理到驱动器连接。

JDBC API的使用驱动程序管理器和数据库特定的驱动程序提供透明的连接到异构数据库。

JDBC驱动程序管理器可确保正确的驱动程序来访问每个数据源。该驱动程序管理器能够支持连接到多个异构数据库的多个并发的驱动程序。

以下是结构图，它显示了驱动程序管理器方面的JDBC驱动程序和Java应用程序的位置：




# 常见的JDBC组件：

JDBC API提供了以下接口和类：

- DriverManager: 这个类管理数据库驱动程序的列表。内容是否符合从Java应用程序使用的通信子协议正确的数据库驱动程序的连接请求。识别JDBC在一定子协议的第一个驱动器将被用来建立数据库连接。
- Driver: 此接口处理与数据库服务器通信。很少直接与驱动程序对象。相反，使用DriverManager中的对象，它管理此类型的对象。它也抽象与驱动程序对象工作相关的详细信息
- Connection : 此接口与接触数据库的所有方法。连接对象表示通信上下文，即，与数据库中的所有的通信是通过唯一的连接对象。
- Statement : 可以使用这个接口创建的对象的SQL语句提交到数据库。一些派生的接口接受除执行存储过程的参数。
- ResultSet: 这些对象保存从数据库后，执行使用Statement对象的SQL查询中检索数据。它作为一个迭代器，让您可以通过移动它的数据。

- SQLException: 这个类处理发生在一个数据库应用程序的任何错误。

## JDBC 4.0软件包

对JDBC4.0，java.sql和javax.sql是主要的包。这是最新版本的JDBC在写这篇教程的时候。它提供的主要类与数据源进行交互。

在这些包中的新功能包括改变在以下几个方面：

- 自动数据库驱动程序加载
- 异常处理的改进
- 增强的BLOB/ CLOB功能
- 连接和语句界面增强
- 国家字符集支持
- SQL ROWID访问
- SQL 2003 XML数据类型支持
- 注释

# JDBC SQL语法 - JDBC教程

结构化查询语言（SQL）是一种标准化的语言，它允许你在数据库上执行操作，如创建项目，读取内容，内容更新和删除条目。

SQL是所有可能会使用几乎任何数据库支持，它允许独立于底层数据库的写入数据库的代码。

本教程给出的SQL，这是一个先决条件，了解JDBC概述。本教程提供了足够的SQL，以便能够创建，读取，更新和删除（通常被称为CRUD操作）从一个数据库中的数据。

有关SQL的详细了解，可以阅读我们的MySQL教程。

## 创建数据库：

CREATE DATABASE语句用于创建一个新的数据库。语法是：

```
SQL> CREATE DATABASE DATABASE_NAME;
```

## 例子：

下面的SQL语句创建一个名为EMP数据库：

```
SQL> CREATE DATABASE EMP;
```

## 删除数据库：

使用DROP DATABASE语句用于删除现有的数据库。语法是：

```
SQL> DROP DATABASE DATABASE_NAME;
```

注意：要创建或删除，应该有数据库服务器上管理员权限的数据库。请注意，删除数据库将所有损失存储在数据库中的数据。

## 创建表：

CREATE TABLE语句用于创建一个新表。语法是：

```
SQL> CREATE TABLE table_name
(
   column_name column_data_type,
   column_name column_data_type,
   column_name column_data_type
   ...
);
```

## 例子：

下面的SQL语句创建一个有四个栏位名为Employees表：

```
SQL> CREATE TABLE Employees
(
   id INT NOT NULL,
   age INT NOT NULL,
   first VARCHAR(255),
   last VARCHAR(255),
   PRIMARY KEY ( id )
);
```

## 删除表：

DROP TABLE语句用于删除现有的表。语法是：

```
SQL> DROP TABLE table_name;
```

## 例子：

下面的SQL语句删除一个名为Employees表：

```
SQL> DROP TABLE Employees;
```

## 插入数据：

语法INSERT类似于以下内容，其中column1, column2，依此类推表示新的数据出现在各列：

```
SQL> INSERT INTO table_name VALUES (column1, column2, ...);
```

## 例子：

下面的SQL INSERT语句中插入先前创建的Employees数据库：

```
SQL> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
```

# SELECT 数据:

SELECT语句用于从数据库中检索数据。该语法的SELECT是：

```
SQL> SELECT column_name, column_name, ...
     FROM table_name
     WHERE conditions;
```

WHERE子句可以使用比较操作符例如=, !=, <, >, <=, >=,以及BETWEEN和LIKE操作符。

## 例子：

下面的SQL语句从Employees表选择age第一个和最后一列名where id =100：

```
SQL> SELECT first, last, age
     FROM Employees
     WHERE id = 100;
```

下面的SQL语句从Employees表，其中第一列选择age，第一列包含 Zara:

```
SQL> SELECT first, last, age
     FROM Employees
     WHERE first LIKE '%Zara%';
```

# UPDATE 数据:

UPDATE语句用于更新数据。UPDATE语法为：

```
SQL> UPDATE table_name
     SET column_name = value, column_name = value, ...
     WHERE conditions;
```

WHERE子句可以使用比较操作符例如=，！=，<，>，<=，和>=，以及BETWEEN和LIKE操作符。

## 例子：

下面的SQL的UPDATE语句更改其ID为100的员工的age列：

```
SQL> UPDATE Employees SET age=20 WHERE id=100;
```

# DELETE 数据:

DELETE语句用来删除表中的数据。语法DELETE是：

```
SQL> DELETE FROM table_name WHERE conditions;
```

WHERE子句可以使用比较操作符例如=，！=，<，>，<=，和>=，以及BETWEEN和LIKE操作符。

## 例子：

下面的SQL DELETE语句删除ID为100的员工的记录：

```
SQL> DELETE FROM Employees WHERE id=100;
```

# JDBC环境设置 - JDBC教程

要开始使用JDBC设置,按照以下所示的步骤开发JDBC环境。以下内容假设Windows平台上。

## 安装Java:

安装J2SE开发工具包5.0（JDK 5.0）下载：Java官方网站。

请确保以下环境变量设置，如下所述：

- **JAVA_HOME**: 此环境变量应该指向安装JDK的目录，例如：C:Program FilesJavajdk1.5.0
- **CLASSPATH**: 此环境变量应已适当的路径设置，如： C:Program FilesJavajdk1.5.0_20jrelib
- **PATH**: 此环境变量应指向适当的JRE bin，如： C:Program FilesJavajre1.5.0_20in.

可能已有这些变量设置了，但只是为了确保在这里是如何检查。

- 进入控制面板，双击系统。如果是Windows XP的用户有可能要打开："性能"=》"维护"，会看到系统图标。
- 转到“高级"选项卡，然后单击"环境变量"。
- 现在，选择所有输入的变量设置正确。

将自动获得了JDBC包java.sql和javax.sql，当安装J2SE开发工具包5.0（JDK 5.0）

## 安装数据库：

将需要当然，最重要的是实际运行的数据库用，可以查询和修改表。

安装数据库是最适合的。可以有很多选择，最常见的有：

- MySQL DB: MySQL是一个开放源码的数据库。可以从这里下载MySQL官方网站，建议下载完整Windows安装。

  此外，下载和安装MySQL管理以及MySQL查询浏览器。这些都是基于GUI的工具，这将使开发更加容易。

  最后，请下载并在一个方便的目录解压缩的MySQL Connector/J（MySQL JDBC驱动程序）。对于本教程的目的，我们将假设已经安装了驱动程序位于C:Program FilesMySQLmysql-connector-java-5.1.8.

相应地设置CLASSPATH变量到C:Program FilesMySQLmysql-connector-java-5.1.8mysql-connector-java-5.1.8-bin.jar. 根据安装的驱动程序版本可能会有所不同。

- PostgreSQL DB: PostgreSQL是一个开放源码的数据库。可以从这里下载PostgreSQL官方网站。

  Postgres安装包含一个名为pgAdmin III一个基于GUI管理工具。 JDBC驱动程序也包括作为安装的一部分。

- Oracle DB: Oracle数据库是Oracle销售的商用数据库。假设有必须的分发介质进行安装。

  Oracle的安装包括一个名为Enterprise Manager中基于GUI的管理工具。 JDBC驱动程序也包括作为安装的一部分。

## 安装数据库驱动程序：

最新的JDK包含JDBC-ODBC桥驱动程序，使大多数开放式数据库连接（ODBC）驱动程序程序员可使用JDBC API。

现在，大多数数据库厂商随数据库的安装提供相应的JDBC驱动程序。所以，不应该担心这部分。

## 设置数据库认证：

在本教程中，我们将使用MySQL数据库。当安装上述任何数据库，它的管理员ID设置为root，并给出规定设置选择的密码。

用root和密码，可以创建另一个用户ID和密码，或者可以使用root和密码在JDBC应用程序中。

有各种不同的数据库操作，如数据库的创建和删除，这将需要管理员ID和密码。

对于JDBC教程的其余部分，我们将使用MySQL数据库 username 作为ID和password 作为密码。

如果没有足够的权限来创建新的用户，那么可以让数据库管理员（DBA）来创建一个用户ID和密码给你。

## 创建数据库：

要创建EMP数据库，请使用下列步骤：

## 步骤1：

打开命令提示符并更改到安装目录，如下所示：

```
C:>
C:>cd Program FilesMySQLin
C:Program FilesMySQLin>
```

注：这取决于的MySQL系统上的安装位置，mysqld.exe的路径可能会有所不同。也可以查看关于如何启动和停止数据库服务器文档。

## 步骤2：

通过执行下面的命令，如果它没有运行启动数据库服务器。

```
C:Program FilesMySQLin>mysqld
C:Program FilesMySQLin>
```

## 步骤3：

通过执行以下命令来创建数据库EMP

```
C:Program FilesMySQLin> mysqladmin create EMP -u root -p
Enter password: ********
C:Program FilesMySQLin>
```

# 创建表

要创建Employees表中EMP的数据库，请执行以下步骤：

## 步骤1：

打开命令提示符并更改到安装目录，如下所示：

```
C:>
C:>cd Program FilesMySQLin
C:Program FilesMySQLin>
```

## 步骤2：

登录数据库，如下所示

```
C:Program FilesMySQLin>mysql -u root -p
Enter password: ********
mysql>
```

## 步骤3：

创建Employee表如下：

```
mysql> use EMP;
mysql> create table Employees
    -> (
    -> id int not null,
    -> age int not null,
    -> first varchar (255),
    -> last varchar (255)
    -> );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

# 创建数据记录

最后，在Employee表中创建一些记录如下：

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)

mysql>
```

关于MySQL数据库的完整认识，学习MySQL教程。

现在，就可以开始使用JDBC尝试。接下来的教程将给予有关JDBC编程的一个样本示例。

# JDBC示例代码 - JDBC教程

本教程提供了如何创建一个简单的JDBC应用程序的示例。演示如何打开一个数据库连接，执行SQL查询，并显示结果。

所有在此模板的例子中提到的步骤，将在本教程的后续章节说明。

## 创建JDBC应用程序：

有下列涉及构建JDBC应用程序的六个步骤：

- 导入数据包 . 需要包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 就可以了.
- 注册**JDBC**驱动程序. 需要初始化驱动程序，可以与数据库打开一个通信通道。
- 打开连接. 需要使用DriverManager.getConnection() 方法创建一个Connection对象，它代表与数据库的物理连接。
- 执行查询 . 需要使用类型声明的对象建立并提交一个SQL语句到数据库。
- 从结果集中提取数据 . 要求使用适当的关于ResultSet.getXXX()方法来检索结果集的数据。
- 清理环境. 需要明确地关闭所有的数据库资源相对依靠JVM的垃圾收集。

## 示例代码：

这个范例的例子可以作为一个模板，在需要建立JDBC应用程序。

基于对环境和数据库安装在前面的章节中做此示例代码已写入。

复制下面的例子FirstExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class FirstExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
```

```
   Connection conn = null;
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();
      String sql;
      sql = "SELECT id, first, last, age FROM Employees";
      ResultSet rs = stmt.executeQuery(sql);

      //STEP 5: Extract data from result set
      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
      //STEP 6: Clean-up environment
      rs.close();
      stmt.close();
      conn.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
```

```
        }//end finally try
    }//end try
    System.out.println("Goodbye!");
}//end main
}//end FirstExample
```

现在来编译上面的例子如下：

```
C:>javac FirstExample.java
C:>
```

当运行FirstExample，它会产生以下结果：

```
C:>java FirstExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
C:>
```

# JDBC驱动类型 - JDBC教程

## 什么是JDBC驱动程序？

JDBC驱动程序实现JDBC API中定义的接口，用于与数据库服务器进行交互。

例如，使用JDBC驱动程序可以打开数据库连接，并通过发送SQL或数据库命令，然后在收到结果与Java进行交互。

java.sql包中附带的JDK包含定义各种类与他们的行为和实际实现在第三方驱动程序。第三方供应商实现了他们的数据库驱动程序的java.sql.Driver接口。

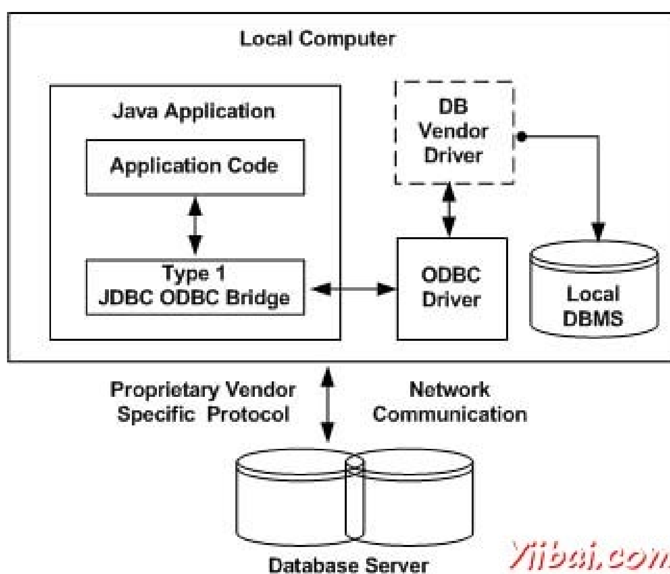
## JDBC驱动程序类型：

JDBC驱动程序实现，因为各种各样的操作系统和Java运行在硬件平台的不同而不同。 Sun已经划分了实现类型分为四大类，类型1，2，3，4，其解释如下：

## 类型1：JDBC-ODBC桥驱动程序：

在类型1驱动程序，一个JDBC桥接器是用来访问安装在每个客户机上的ODBC驱动程序。使用ODBC，需要配置系统数据源名称（DSN），表示在目标数据库上。

当Java刚出来时，这是一个有用的驱动程序，因为大多数的数据库只支持ODBC访问，但现在建议使用此类型的驱动程序仅用于实验用途或在没有其他选择的情况。

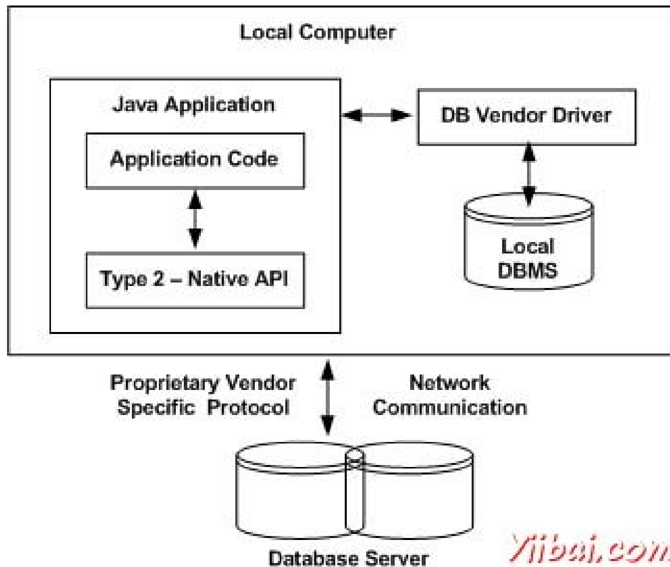



自带JDK 1.2中的JDBC-ODBC桥是这类驱动程序的一个很好的例子。

## 类型2：JDBC-Native API调用：

在类型2驱动程序，JDBC API调用转换成原生的C / C+ + API调用都有它独特的数据库。这些通常由数据库厂商提供，并以相同的方式将JDBC-ODBC桥驱动程序使用，特定供应商的驱动程序必须安装在每台客户机上。

如果改变了数据库，那么必须改变本机API，因为它是具体到一个数据库，他们大多是过时了，但现在可以实现一些速度增加了类型2驱动程序，因为它消除了数据库的开销。

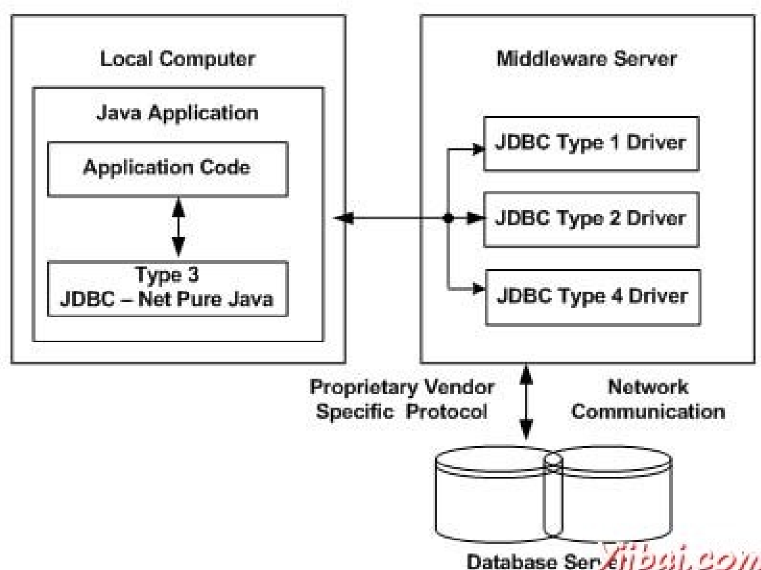



Oracle调用接口（OCI）驱动程序是一个类型2驱动程序的一个示例。

## 类型3：JDBC网络纯Java：

在类型3驱动程序，一个三层的方法来访问数据库。在JDBC客户端使用标准的网络套接字与中间件应用服务器进行通信。套接字的相关信息，然后由中间件应用服务器进入由DBMS所需要的的调用格式转换，并转发到数据库服务器。

这种驱动器是非常灵活的，因为它不需要安装在客户端上的代码和一个单一的驱动器实际上可以提供访问多个数据库。



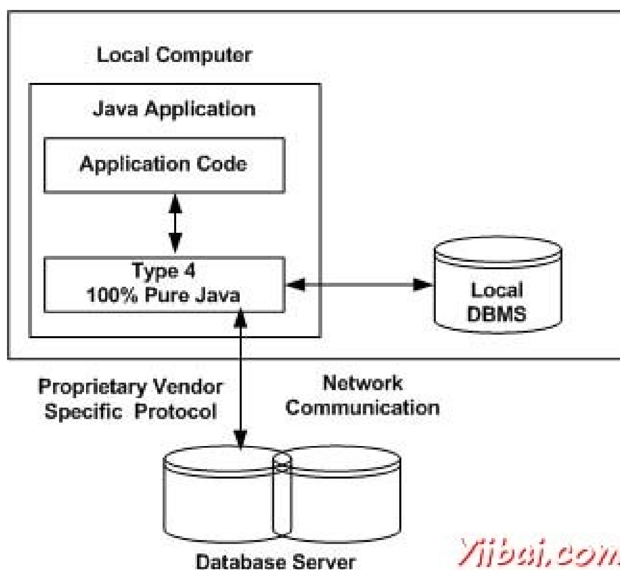
可以将应用服务器作为一个JDBC“代理”，这意味着它会调用客户端应用程序。因此，需要应用服务器的配置，以有效地使用此驱动程序类型的一些知识。

应用服务器可能使用类型1，2，或4驱动程序与数据库进行通信，了解细微之处将证明是有益的。

## 类型4：100%纯Java：

在一个类型4驱动程序，直接与供应商的数据库进行通信，通过socket连接一个纯粹的基于Java的驱动程序。这是可用于数据库的最高性能的驱动程序，并且通常由供应商本身提供。

这种驱动器是非常灵活的，不需要在客户端或服务器上安装特殊的软件。此外，这些驱动程序可以动态下载。




MySQL的Connector/J的驱动程序是一个类型4驱动程序。因为他们的网络协议的专有性的，数据库厂商通常提供类型4的驱动程序。

## 其中驱动器应使用？

如果正在访问一个类型的数据库，如Oracle，Sybase或IBM，首选驱动程序是类型4。

如果Java应用程序同时访问多个数据库类型，类型3是首选的驱动程序。

第2类驱动程序是在情况下：类型3或类型4驱动程序还没有提供数据库非常有用。

类型1驱动程序不被认为是部署级别的驱动程序，它通常仅用于开发和测试目的。

# JDBC连接数据库 - JDBC教程

在安装相应的驱动程序后，现在是时候建立使用JDBC的数据库连接。

涉及到建立一个JDBC连接的编程是相当简单的。下面是这些简单的四个步骤：

- 导入**JDBC**包： 添加import语句到Java程序导入所需的类在Java代码中。
- 注册**JDBC**驱动程序：这一步会导致JVM加载所需的驱动程序实现到内存中，因此它可以实现JDBC请求。
- 数据库**URL**制定：这是创建格式正确的地址指向到要连接的数据库。
- 创建连接对象：最后，代码调用DriverManager对象的getConnection()方法来建立实际的数据库连接。

## 导入JDBC包：

import 语句告诉Java编译器在哪里可以找到在代码中引用，并放置在您的源代码最开始的类。

使用标准的JDBC包，它允许选择，插入，更新和SQL表中删除数据，添加以下进口到您的源代码：

```
import java.sql.* ;  // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger support
```

## 注册JDBC驱动程序：

使用它之前，必须注册你的驱动程序在程序。注册驱动程序是由Oracle驱动程序的类文件被加载到内存中以便它可以被用作JDBC接口的实现过程。

需要做这个注册只能在你的程序一次。可以通过以下两种方式之一注册一个驱动程序。

## 方法（I）- Class.forName()：

注册一个驱动程序中最常用的方法是使用Java的Class.forName()方法来动态加载驱动程序的类文件到内存中，它会自动将其注册。这种方法是可取的，因为它允许使驱动注册配置，便于携带。

下面的示例使用Class.forName()来注册Oracle驱动程序：

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

可以使用getInstance()方法来解决不兼容的JVM，但要编写了两个额外的例外情况如下：

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
catch(IllegalAccessException ex) {
   System.out.println("Error: access problem while loading!");
   System.exit(2);
catch(InstantiationException ex) {
   System.out.println("Error: unable to instantiate driver!");
   System.exit(3);
}
```

## 方法（二） - DriverManager.registerDriver()：

可以用它来注册一个驱动程序的第二种方法是使用staticDriverManager.registerDriver()方法。

应该，如果使用的是不兼容的JDK JVM，比如微软提供一个使用registerDriver()方法。

下面的示例使用registerDriver()来注册Oracle驱动程序：

```
try {
   Driver myDriver = new oracle.jdbc.driver.OracleDriver();
   DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

# 数据库URL制定：

当加载的驱动程序，可以建立程序中使用DriverManager.getConnection()方法的连接。为方便参考，让列出了三个重载DriverManager.getConnection()方法：

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

在这里，每个表单需要一个数据库URL。数据库的URL是指向数据库地址。

制定一个数据库URL是大多数用在建立连接相关。

下表列出了下来流行的JDBC驱动程序名和数据库的URL。

| RDBMS | JDBC驱动程序的名称 | URL 格式 |
|---|---|---|
| MySQL | com.mysql.jdbc.Driver | **jdbc:mysql://**hostname/ databaseName |
| ORACLE | oracle.jdbc.driver.OracleDriver | **jdbc:oracle:thin:@**hostname:p Number:databaseName |
| DB2 | COM.ibm.db2.jdbc.net.DB2Driver | **jdbc:db2:**hostname:port Number/databaseName |
| Sybase | com.sybase.jdbc.SybDriver | **jdbc:sybase:Tds:**hostname: p Number/databaseName |

以URL格式所有高亮的部分是静态的，需要改变只剩余部分按照数据库设置。

# 创建连接对象：

# 使用数据库URL的用户名和密码：

下面三种形式DriverManager.getConnection()方法来创建一个连接对象。getConnection()最常用形式要求传递一个数据库URL，用户名 username和密码 password：

对URL数据库部分databaseName的值：假设使用的是Oracle的瘦驱动程序，需要指定一个主机：端口。

假设有一台主机TCP/IP地址192.0.0.1 以及主机名和Oracle监听器被配置为在端口1521，数据库名称是EMP，然后完整的数据库URL是：

```
jdbc:oracle:thin:@amrood:1521:EMP
```

现在，必须调用适当的用户名和密码以及getConnection()方法来获得一个Connection对象，如下所示：

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

## 只使用一个数据库URL：

第二种形式 DriverManager.getConnection()方法只需要一个数据库URL：

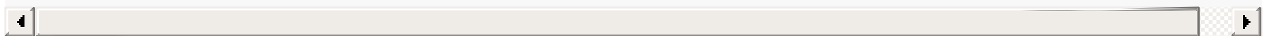
```
DriverManager.getConnection(String url);
```

然而，在这种情况下，数据库的URL，包括用户名和密码，并具有以下的一般形式：

```
jdbc:oracle:driver:username/password@database
```

所以上面的连接可以创建如下：

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```



## 使用数据库的URL和一个Properties对象：

第三种形式DriverManager.getConnection()方法需要一个数据库URL和一个Properties对象：

```
DriverManager.getConnection(String url, Properties info);
```

Properties对象，保存一组关键字 - 值对。它被用来调用getConnection()方法时驱动程序属性传递给驱动程序。

为了使通过前面的例子中所做的相同的连接，使用下面的代码：

```
import java.util.*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);
```

# 关闭JDBC连接：

在JDBC程序的结束，它明确要求关闭所有的连接到数据库，以结束每个数据库会话。但是，如果忘了，Java垃圾收集器会关闭连接时，它会清除陈旧的对象。

依托垃圾收集，特别是在数据库编程，是非常差的编程习惯。应该总是在关闭与连接对象关联的close()方法连接的习惯。

为了确保连接被关闭，可以在代码中的finally块执行。 finally块都会执行，不管是否发生或也不例外。

要关闭上面打开的连接，应该调用close()方法，如下所示：

```
conn.close();
```

显式地关闭连接DBMS节约资源。

为了更好地理解，建议看看JDBC - 示例代码。

# JDBC Statements,PreparedStatement和CallableStatement - JDBC教程

一旦获得一个连接，我们可以与数据库进行交互。在JDBC Statement, CallableStatement 和 PreparedStatement 接口定义的方法和属性，使可以发送SQL或PL/SQL命令和从数据库接收数据。

它们还定义方法，帮助Java和数据库使用SQL数据类型之间转换数据的差异。

下表提供了每个接口的用途概要，了解决定使用哪个接口？

| 接口 | 推荐使用 |
| --- | --- |
| Statement | 使用通用访问数据库。当在运行时使用静态SQL语句。Statement接口不能接受的参数。 |
| PreparedStatement | 当计划多次使用SQL语句。 那么可以PreparedStatement接口接收在运行时输入参数。 |
| CallableStatement | 当要访问数据库中的存储过程中使用。CallableStatement对象的接口还可以接受运行时输入参数。 |

## Statement 对象:

## 创建Statement对象:

在可以使用Statement对象执行SQL语句，需要使用Connection对象的createStatement( )方法创建一个，如下面的示例所示：

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

一旦创建了一个Statement对象，然后可以用它来与它的三个执行方法之一执行SQL语句。

- **boolean execute(String SQL) :** 如果ResultSet对象可以被检索返回布尔值true，否则返回false。使用这个方法来执行SQL DDL语句，或当需要使用真正的动态SQL。
- **int executeUpdate(String SQL) :** 返回受影响的SQL语句执行的行的数目。使用此方法来执行，而希望得到一些受影响的行的SQL语句 - 例如，INSERT，UPDATE或DELETE语句。
- **ResultSet executeQuery(String SQL) :** 返回ResultSet对象。当希望得到一个结果集使用此方法，就像使用一个SELECT语句。

## 关闭 Statement 对象:

正如关闭一个Connection对象来保存数据库资源，出于同样的原因，也应该关闭Statement对象。

close()方法简单的调用将完成这项工作。如果关闭了Connection对象首先它会关闭Statement对象也是如此。然而，应该始终明确关闭Statement对象，以确保正确的清除。

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   stmt.close();
}
```

为了更好地理解，建议研究和学习 Statement 实例代码.

## PreparedStatement 对象:

PreparedStatement接口扩展了Statement接口，让过一个通用的Statement对象增加几个高级功能。

statement 提供动态参数的灵活性。

## 创建PreparedStatement 对象:

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

在JDBC中所有的参数都被代表？符号，这是已知的参数标记。在执行SQL语句之前，必须提供值的每一个参数。

setXXX()方法将值绑定到参数，其中XXX表示希望绑定到输入参数值的Java数据类型。如果忘了提供值，将收到一个SQLException。

每个参数标记是由它的序号位置引用。第一标记表示位置1，下一个位置为2 等等。这种方法不同于Java数组索引，以0开始。

所有的Statement对象的方法来与数据库交互(a) execute(), (b) executeQuery(), 及 (c) executeUpdate() 也与PreparedStatement对象的工作。然而，该方法被修改为使用SQL语句，可以利用输入的参数。

# 关闭**PreparedStatement**对象:

正如关闭Statement对象，出于同样的原因，也应该关闭的PreparedStatement对象。

close()方法简单的调用将完成这项工作。如果关闭了Connection对象首先它会关闭PreparedStatement对象。然而，应该始终明确关闭PreparedStatement对象，以确保正确的清除。

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   pstmt.close();
}
```

为了更好地理解，建议学习Prepare 例子代码.

## CallableStatement 对象:

正如一个Connection对象创建Statement和PreparedStatement对象，它也创造了CallableStatement对象这将被用来执行调用数据库存储过程。

## 创建CallableStatement 对象:

假设，需要执行以下Oracle存储过程：

```
CREATE OR REPLACE PROCEDURE getEmpName
   (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END;
```

注：上面的存储过程已写入用于Oracle，但我们正在使用MySQL数据库，以便让我们写相同的存储过程对于MySQL如下，以EMP数据库中创建它：

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

三种类型的参数有：IN，OUT和INOUT。 PreparedStatement对象只使用IN参数。CallableStatement对象可以使用所有三个。

这里是每个的定义：

| 参数 | 描述 |
| --- | --- |
| IN | 它的值是在创建SQL语句时未知的参数。将值绑定到与setXXX()方法的参数。 |
| OUT | 其值由它返回的SQL语句提供的参数。从OUT参数的getXXX()方法检索值。 |
| INOUT | 同时提供输入和输出值的参数。绑定的setXXX()方法的变量，并使用getXXX()方法检索值。 |

下面的代码片段显示了如何使用该Connection.prepareCall()方法实例化基于上述存储过程CallableStatement对象：

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

String变量的SQL表示存储过程，使用参数占位符。

使用CallableStatement对象是就像使用PreparedStatement对象。必须将值绑定到所有的参数执行该语句之前，否则将收到一个SQLException。

如果有IN参数，只要按照适用于PreparedStatement对象相同的规则和技巧;使用对应于要绑定的Java数据类型的setXXX()方法。

当使用OUT和INOUT参数就必须采用额外的CallableStatement及registerOutParameter()方法。registerOutParameter()方法JDBC数据类型绑定到数据类型的存储过程应返回。

一旦调用存储过程，用适当的getXXX()方法的输出参数检索值。这种方法投射SQL类型的值检索到Java数据类型。

## 关闭CallableStatement 对象：

正如关闭其他Statement对象，出于同样的原因，也应该关闭CallableStatement对象。

close()方法简单的调用将完成这项工作。如果关闭了Connection对象首先它会关闭CallableStatement对象为好。然而，应该始终明确关闭CallableStatement对象，以确保正确的清除。

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

为了更好地理解，建议学习 Callable实例代码。

# JDBC PreparedStatement对象实例 - JDBC教程

以下是这使得使用PreparedStatement以及打开和关闭statments的例子：

基于对环境和数据库安装在前面的章节中进行这个范例程式码已被写入。

复制下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   PreparedStatement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      String sql = "UPDATE Employees set age=? WHERE id=?";
      stmt = conn.prepareStatement(sql);

      //Bind values into the parameters.
      stmt.setInt(1, 35);  // This would set age
      stmt.setInt(2, 102); // This would set ID

      // Let us update age of the record with ID = 102;
      int rows = stmt.executeUpdate();
      System.out.println("Rows impacted : " + rows );

      // Let us select all the records and display them.
      sql = "SELECT id, first, last, age FROM Employees";
      ResultSet rs = stmt.executeQuery(sql);

      //STEP 5: Extract data from result set
      while(rs.next()){
```

```
            //Retrieve by column name
            int id  = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last");

            //Display values
            System.out.print("ID: " + id);
            System.out.print(", Age: " + age);
            System.out.print(", First: " + first);
            System.out.println(", Last: " + last);
         }
         //STEP 6: Clean-up environment
         rs.close();
         stmt.close();
         conn.close();
      }catch(SQLException se){
         //Handle errors for JDBC
         se.printStackTrace();
      }catch(Exception e){
         //Handle errors for Class.forName
         e.printStackTrace();
      }finally{
         //finally block used to close resources
         try{
            if(stmt!=null)
               stmt.close();
         }catch(SQLException se2){
         }// nothing we can do
         try{
            if(conn!=null)
               conn.close();
         }catch(SQLException se){
            se.printStackTrace();
         }//end finally try
      }//end try
      System.out.println("Goodbye!");
   }//end main
   }//end JDBCExample
```

现在来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to database...
Creating statement...
Rows impacted : 1
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 35, First: Zaid, Last: Khan
ID: 103, Age: 30, First: Sumit, Last: Mittal
Goodbye!
C:>
```

# JDBC CallableStatement对象实例 - JDBC教程

下面是利用CallableStatement连同下列getEmpName()的MySQL存储过程的例子：

请确定已经在EMP数据库中创建该存储过程。可以使用MySQL查询浏览器来完成它。

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

基于对环境和数据库安装在前面的章节中进行，这个范例程式码已被写入。

复制下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   CallableStatement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //STEP 4: Execute a query
```

```
      System.out.println("Creating statement...");
      String sql = "{call getEmpName (?, ?)}";
      stmt = conn.prepareCall(sql);

      //Bind IN parameter first, then bind OUT parameter
      int empID = 102;
      stmt.setInt(1, empID); // This would set ID as 102
      // Because second parameter is OUT so register it
      stmt.registerOutParameter(2, java.sql.Types.VARCHAR);

      //Use execute method to run stored procedure.
      System.out.println("Executing stored procedure..." );
      stmt.execute();

      //Retrieve employee name with getXXX method
      String empName = stmt.getString(2);
      System.out.println("Emp Name with ID:" +
               empID + " is " + empName);
      stmt.close();
      conn.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to database...
Creating statement...
Executing stored procedure...
Emp Name with ID:102 is Zaid
Goodbye!
C:>
```

# JDBC结果集Result/Sets - JDBC教程

从数据库查询读取数据的SQL语句返回的结果集的数据。 SELECT语句的标准方法来选择从一个数据库中的行并查来在一个结果集。java.sql.ResultSet接口表示一个数据库查询的结果集。

一个ResultSet对象维护一个游标指向当前行的结果集。术语“结果集”是指包含在ResultSet对象中的行和列的数据。

ResultSet接口的方法可分为三类：

- 导航方法：用于移动光标.
- 获取方法：用于查看当前行的光标所指向的列中的数据。
- 更新方法：用于更新当前行的列中的数据。这些更新然后可以在基础数据库中，以及更新。

将光标移动基于ResultSet的属性。所创建生成ResultSet相应的声明时，这些属性被指定。

JDBC提供下列连接方法来创建所需的ResultSet语句：

- *createStatement(int RSType, int RSConcurrency);*
- *prepareStatement(String SQL, int RSType, int RSConcurrency);*
- *prepareCall(String sql, int RSType, int RSConcurrency);*

第一个参数表示ResultSet对象的类型，第二个参数是2的ResultSet常量，用于指定一个结果集是否为只读或可更新之一。

## ResultSet的类型：

可能的RSType如下，如果不指定ResultSet类型，将自动获得一个是TYPE_FORWARD_ONLY。

| Type | 描述 |
|---|---|
| ResultSet.TYPE_FORWARD_ONLY | 游标只能向前移动的结果集。 |
| ResultSet.TYPE_SCROLL_INSENSITIVE | 游标可以向前和向后滚动，结果集不是别人向创建结果集后发生的数据库更改敏感。 |
| ResultSet.TYPE_SCROLL_SENSITIVE. | 游标可以向前和向后滚动，结果集是别人向创建结果集后发生的数据库更改敏感。 |

# 并发性的ResultSet：

可能的RSConcurrency如下，如果不指定任何并发类型，将自动获得一个为CONCUR_READ_ONLY。

| 并发 | 描述 |
|---|---|
| ResultSet.CONCUR_READ_ONLY | 创建结果集只读。这是默认的 |
| ResultSet.CONCUR_UPDATABLE | 创建一个可更新的结果集。 |

到目前为止已写入的例子可以写成如下的初始化一个Statement对象来创建一个只进，只读的ResultSet对象：

```
try {
   Statement stmt = conn.createStatement(
                          ResultSet.TYPE_FORWARD_ONLY,
                          ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) {
   ....
}
finally {
   ....
}
```

# 导航结果集：

有几种方法在ResultSet接口涉及移动光标，包括：

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **public void beforeFirst() throws SQLException** 将光标移动到正好位于第一行之前 |
| 2 | **public void afterLast() throws SQLException** 将光标移动到刚刚结束的最后一行 |
| 3 | **public boolean first() throws SQLException** 将光标移动到第一行 |
| 4 | **public void last() throws SQLException** 将光标移动到最后一行。 |
| 5 | **public boolean absolute(int row) throws SQLException** 将光标移动到指定的行 |
| 6 | **public boolean relative(int row) throws SQLException** 从它目前所指向向前或向后移动光标行的给定数量。 |
| 7 | **public boolean previous() throws SQLException** 将光标移动到上一行。上一行关闭的结果集此方法返回false |
| 8 | **public boolean next() throws SQLException** 将光标移动到下一行。如果没有更多的行结果集中的此方法返回false |
| 9 | **public int getRow() throws SQLException** 返回的行号，该光标指向的行. |
| 10 | **public void moveToInsertRow() throws SQLException** 将光标移动到一个特殊的行，可以用来插入新行插入到数据库中的结果集。当前光标位置被记住. |
| 11 | **public void moveToCurrentRow() throws SQLException** 移动光标返回到当前行，如果光标在当前插入行，否则，这个方法不执行任何操作 |

为了更好地理解，建议学习导航示例代码

## 查看结果集：

ResultSet接口中含有几十种用于获取当前行的数据的方法。

有一个get方法为每个可能的数据类型，并且每个get方法有两个版本:

- 即需要在一个列名。
- 即需要在列中索引。

例如，如果有兴趣查看的列包含一个整数，需要使用ResultSet调用getInt()方法之一：

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **public int getInt(String columnName) throws SQLException** 返回整数的当前行中名为ColumnName列 |
| 2 | **public int getInt(int columnIndex) throws SQLException** 返回整数的当前行中指定列的索引。列索引从1开始，意味着一个行的第一列是1，行的第二列是2，依此类推。 |

与此类似的还有get方法在ResultSet接口为每个八个Java原始类型，以及常见的类型比如java.lang.String，java.lang.Object和java.net.URL

也有用于获取SQL数据类型java.sql.Date，java.sql.Time，java.sql.Timestamp，java.sql.Clob，java.sql.Blob中的方法。检查有关使用这些SQL数据类型的详细信息的文档。

为了更好地理解，建议学习 查看 - 示例代码.

## 更新的结果集：

ResultSet接口中包含的更新方法用于更新的结果集的数据的集合。

由于get方法，有两种更新方法为每种数据类型：

- 即需要在一个列名。
- 即需要在列中索引。

例如，要更新一个结果集的当前行的String列，可以使用下面的updateString（）方法之一：

| S.N. | 方法& 描述 |
|---|---|
| 1 | **public void updateString(int columnIndex, String s) throws SQLException** 指定列中的字符串更改为s的值。 |
| 2 | **public void updateString(String columnName, String s) throws SQLException** 类似于前面的方法，不同之处在于由它的名称，而不是它的索引指定的列。 |

有更新方法八个原始数据类型，以及字符串，对象，URL，并在java.sql包中的SQL数据类型。

更新结果集中的行改变当前行的列中的ResultSet对象，而不是基础数据库中。要更新更改数据库中的一行，需要调用下面的方法之一。

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **public void updateRow()** 通过更新数据库中相应的行更新当前行。 |
| 2 | **public void deleteRow()** 从数据库中删除当前行 |
| 3 | **public void refreshRow()** 刷新在结果集的数据，以反映最新变化在数据库中。 |
| 4 | **public void cancelRowUpdates()** 取消所做的当前行的任何更新。 |
| 5 | **public void insertRow()** 插入一行到数据库中。当光标指向插入行此方法只能被调用。 |

为了更好地理解，建议学习更新示例代码。

# JDBC数据类型 - JDBC教程

JDBC驱动程序将其发送到数据库之前的Java数据类型转换为相应的JDBC类型。它采用了默认的映射对于大多数数据类型。例如，一个Java整型转换为SQL INTEGER。创建默认映射提供的驱动程序之间的一致性。

下表总结了Java数据类型转换为当调用PreparedStatement中的setXXX()方法或CallableStatement对象或ResultSet.updateXXX()方法的默认JDBC数据类型。

| SQL | JDBC/Java | setXXX | updateXXX |
|---|---|---|---|
| VARCHAR | java.lang.String | setString | updateString |
| CHAR | java.lang.String | setString | updateString |
| LONGVARCHAR | java.lang.String | setString | updateString |
| BIT | boolean | setBoolean | updateBoolean |
| NUMERIC | java.math.BigDecimal | setBigDecimal | updateBigDecim |
| TINYINT | byte | setByte | updateByte |
| SMALLINT | short | setShort | updateShort |
| INTEGER | int | setInt | updateInt |
| BIGINT | long | setLong | updateLong |
| REAL | float | setFloat | updateFloat |
| FLOAT | float | setFloat | updateFloat |
| DOUBLE | double | setDouble | updateDouble |
| VARBINARY | byte[ ] | setBytes | updateBytes |
| BINARY | byte[ ] | setBytes | updateBytes |
| DATE | java.sql.Date | setDate | updateDate |
| TIME | java.sql.Time | setTime | updateTime |
| TIMESTAMP | java.sql.Timestamp | setTimestamp | updateTimestam |
| CLOB | java.sql.Clob | setClob | updateClob |
| BLOB | java.sql.Blob | setBlob | updateBlob |
| ARRAY | java.sql.Array | setARRAY | updateARRAY |
| REF | java.sql.Ref | SetRef | updateRef |
| STRUCT | java.sql.Struct | SetStruct | updateStruct |

JDBC3.0增强了对BLOB，CLOB，ARRAY和REF数据类型的支持。 ResultSet 对象现在有UPDATEBLOB()，updateCLOB()，updateArray()，和updateRef()方法，可以直接操作服务器上的相应数据。

setXXX()和updateXXX()方法能够将特定Java类型转换为特定的JDBC数据类型。该方法setObject()和updateObject()，能够将几乎任何Java类型映射到JDBC数据类型。

ResultSet对象为每个数据类型来检索列值对应的getXXX()方法。每个方法可用于与列名或由它的序数位置。

| SQL | JDBC/Java | setXXX | getXXX |
|---|---|---|---|
| VARCHAR | java.lang.String | setString | getString |
| CHAR | java.lang.String | setString | getString |
| LONGVARCHAR | java.lang.String | setString | getString |
| BIT | boolean | setBoolean | getBoolean |
| NUMERIC | java.math.BigDecimal | setBigDecimal | getBigDecimal |
| TINYINT | byte | setByte | getByte |
| SMALLINT | short | setShort | getShort |
| INTEGER | int | setInt | getInt |
| BIGINT | long | setLong | getLong |
| REAL | float | setFloat | getFloat |
| FLOAT | float | setFloat | getFloat |
| DOUBLE | double | setDouble | getDouble |
| VARBINARY | byte[ ] | setBytes | getBytes |
| BINARY | byte[ ] | setBytes | getBytes |
| DATE | java.sql.Date | setDate | getDate |
| TIME | java.sql.Time | setTime | getTime |
| TIMESTAMP | java.sql.Timestamp | setTimestamp | getTimestamp |
| CLOB | java.sql.Clob | setClob | getClob |
| BLOB | java.sql.Blob | setBlob | getBlob |
| ARRAY | java.sql.Array | setARRAY | getARRAY |
| REF | java.sql.Ref | SetRef | getRef |
| STRUCT | java.sql.Struct | SetStruct | getStruct |

# 日期和时间数据类型：

java.sql.Date中的类映射到SQL DATE类型，以及java.sql.Time和java.sql.Timestamp类映射分别到SQL TIME和SQL TIMESTAMP数据类型。

下面的例子显示了日期和时间格式类标准的Java日期和时间值相匹配的SQL数据类型的要求。

```
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;
import java.util.*;

public class SqlDateTime {
   public static void main(String[] args) {
      //Get standard date and time
      java.util.Date javaDate = new java.util.Date();
      long javaTime = javaDate.getTime();
      System.out.println("The Java Date is:" +
             javaDate.toString());

      //Get and display SQL DATE
      java.sql.Date sqlDate = new java.sql.Date(javaTime);
      System.out.println("The SQL DATE is: " +
             sqlDate.toString());

      //Get and display SQL TIME
      java.sql.Time sqlTime = new java.sql.Time(javaTime);
      System.out.println("The SQL TIME is: " +
             sqlTime.toString());
      //Get and display SQL TIMESTAMP
      java.sql.Timestamp sqlTimestamp =
      new java.sql.Timestamp(javaTime);
      System.out.println("The SQL TIMESTAMP is: " +
             sqlTimestamp.toString());
     }//end main
}//end SqlDateTime
```

现在让我们来编译上面的例子如下：

```
C:>javac SqlDateTime.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java SqlDateTime
The Java Date is:Tue Aug 18 13:46:02 GMT+04:00 2009
The SQL DATE is: 2009-08-18
The SQL TIME is: 13:46:02
The SQL TIMESTAMP is: 2009-08-18 13:46:02.828
C:>
```

## 处理NULL值：

SQL使用NULL值和Java使用null是不同的概念。那么，如何处理Java中的SQL NULL值？有三种策略可以使用：

- 避免使用返回原始数据类型的getXXX()方法。
- 使用包装类的基本数据类型，并使用ResultSet对象的wasNull()方法来测试是否是收到getXXX()方法返回的值，包装类变量应该被设置为null。
- 使用原始数据类型和ResultSet对象的wasNull()方法来测试是否是收到getXXX()方法返回的值的原始变量应设置为你已经选择代表一个NULL可接受的值。

下面是一个例子来处理NULL值：

```
Statement stmt = conn.createStatement( );
String sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);

int id = rs.getInt(1);
if( rs.wasNull( ) ) {
   id = 0;
}
```

# JDBC事务 - JDBC教程

如果JDBC连接是在自动提交模式下，它在默认情况下，那么每个SQL语句都是在其完成时提交到数据库。

这可能是对简单的应用程序，但有三个原因，你可能想关闭自动提交和管理自己的事务：

- 为了提高性能
- 为了保持业务流程的完整性
- 使用分布式事务

若要控制事务，以及何时更改应用到数据库。它把单个SQL语句或一组SQL语句作为一个逻辑单元，而且如果任何语句失败，整个事务失败。

若要启用，而不是JDBC驱动程序默认使用auto-commit模式手动事务支持，使用Connection对象的的setAutoCommit()方法。如果传递一个布尔值false到setAutoCommit()，关闭自动提交。可以传递一个布尔值true将其重新打开。

例如，如果有一个名为conn Connection对象，以下代码来关闭自动提交：

```
conn.setAutoCommit(false);
```

## 提交和回滚

一旦已经完成了变化，要提交更改，然后调用commit（在连接对象）方法，如下所示：

```
conn.commit( );
```

否则回滚更新对数据库所做的使用命名连接conn，使用下面的代码：

```
conn.rollback( );
```

下面的例子演示了如何使用一个提交和回滚对象：

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   String SQL = "INSERT INTO Employees  " +
                "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees  " +
                "VALUES (107, 22, 'Sita', 'Singh')";
   stmt.executeUpdate(SQL);
   // If there is no error.
   conn.commit();
}catch(SQLException se){
   // If there is any error.
   conn.rollback();
}
```

在这种情况下没有上述INSERT语句会成功，一切都将被回滚。

为了更好地理解，建议学习事务提交实例代码.

# 使用保存点：

新的JDBC3.0保存点的接口提供了额外的事务控制。他们的环境中，如Oracle的PL/ SQL中的大多数现代的DBMS支持保存点。

当设置一个保存点在事务中定义一个逻辑回滚点。如果发生错误，过去一个保存点，则可以使用rollback方法来撤消要么所有的改变或仅保存点之后所做的更改。

Connection对象有两个新的方法，可帮助管理保存点：

- setSavepoint(String savepointName): 定义了一个新的保存点。它也返回一个Savepoint 对象。
- releaseSavepoint(Savepoint savepointName): 删除一个保存点。请注意，它需要一个Savepoint 对象作为参数。这个对象通常是由setSavepoint()方法生成一个保存点。

有一个rollback ( String savepointName ) 方法回滚工作到指定的保存点。

下面的例子演示如何使用Savepoint对象：

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   //set a Savepoint
   Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
   String SQL = "INSERT INTO Employees " +
                "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees " +
                "VALUES (107, 22, 'Sita', 'Tez')";
   stmt.executeUpdate(SQL);
   // If there is no error, commit the changes.
   conn.commit();

}catch(SQLException se){
   // If there is any error.
   conn.rollback(savepoint1);
}
```

在这种情况下没有上述INSERT语句会成功，一切都将被回滚。

为了更好地理解，建议学习保存点实例代码.

# JDBC异常处理 - JDBC教程

异常处理，可以处理在一个受控制的方式异常情况，如程序定义的错误。

当异常情况发生时，将引发异常。抛出这个词意味着当前执行的程序停止，并控制被重定向到最近的适用的catch子句。如果没有适用 catch 子句存在，那么程序的执行结束。

JDBC的异常处理非常类似于Java Excpetion处理，但对于JDBC，最常见的异常处理的是 **java.sql.SQLException.**

## SQLException 方法:

SqlException异常可以在驱动程序和数据库出现在这两种。当出现这样的异常时，抛出：SQLException类型的对象将被传递到catch子句。

传递的SQLException对象具有可用于检索有关异常的附加信息下面的方法：

| 方法 | 描述 |
|---|---|
| getErrorCode( ) | 获取与异常关联的错误号。 |
| getMessage( ) | 获取JDBC驱动程序的错误消息由驱动程序处理错误或获取Oracle错误号和消息的一个数据库错误。 |
| getSQLState( ) | 获取XOPEN SQLSTATE字符串。对于JDBC驱动程序的错误，没有有用的信息从该方法返回。对于一个数据库错误，则返回五位XOPEN SQLSTATE代码。这种方法可以返回null。 |
| getNextException( ) | 获取异常链的下一个Exception对象。 |
| printStackTrace( ) | 打印当前的异常，或者抛出，其回溯到标准错误流。 |
| printStackTrace(PrintStream s) | 打印此抛出，其回溯到指定的打印流。 |
| printStackTrace(PrintWriter w) | 打印此抛出，其回溯到指定的打印写入。 |

通过利用可从Exception对象捕获异常的信息，并适当地继续运行程序。这里是一个try块的一般形式为：

```
try {
   // Your risky code goes between these curly braces!!!
}
catch(Exception ex) {
   // Your exception handling code goes between these
   // curly braces, similar to the exception clause
   // in a PL/SQL block.
}
finally {
   // Your must-always-be-executed code goes between these
   // curly braces. Like closing database connection.
}
```

# 例如：

学习下面的代码示例来了解试试 **try....catch...finally** 块的使用。

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      Statement stmt = conn.createStatement();
      String sql;
      sql = "SELECT id, first, last, age FROM Employees";
      ResultSet rs = stmt.executeQuery(sql);

      //STEP 5: Extract data from result set
      while(rs.next()){
         //Retrieve by column name
```

```
            int id  = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last");

            //Display values
            System.out.print("ID: " + id);
            System.out.print(", Age: " + age);
            System.out.print(", First: " + first);
            System.out.println(", Last: " + last);
         }
         //STEP 6: Clean-up environment
         rs.close();
         stmt.close();
         conn.close();
      }catch(SQLException se){
         //Handle errors for JDBC
         se.printStackTrace();
      }catch(Exception e){
         //Handle errors for Class.forName
         e.printStackTrace();
      }finally{
         //finally block used to close resources
         try{
            if(conn!=null)
               conn.close();
         }catch(SQLException se){
            se.printStackTrace();
         }//end finally try
      }//end try
      System.out.println("Goodbye!");
   }//end main
   }//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，如果没有问题它会产生以下结果，否则相应的错误将被捕获并会显示错误消息：

```
C:>java JDBCExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
C:>
```

试试上面的例子中通过传递错误的数据库名称或错误的用户名或密码，并检查结果。

# JDBC批量处理 - JDBC教程

批处理允许将相关的SQL语句组合成一个批处理和一个调用数据库提交。

当一次发送多个SQL语句到数据库，可以减少通信开销的数额，从而提高了性能。

- JDBC驱动程序不需要支持此功能。应该使用DatabaseMetaData.supportsBatchUpdates()方法来确定目标数据库支持批量更新处理。如果你的JDBC驱动程序支持此功能的方法返回true。
- 声明addBatch()方法，PreparedStatement和CallableStatement用于各个语句添加到批处理。executeBatch()将用于启动所有组合在一起的语句的执行。
- executeBatch()将返回一个整数数组，数组中的每个元素代表了各自的更新语句的更新计数。
- 可以添加语句批量处理，可以用theclearBatch()方法删除它们。此方法删除所有已添加的addBatch()方法的语句。但是，不能有选择性地选择要删除的语句。

## 批处理和Statement对象：

下面是步骤，使用批处理使用说明书对象的典型顺序：

- 使用createStatement()方法创建一个Statement对象。
- 设置使用自动提交为false，使用 setAutoCommit().
- 添加任意多个到批量使用addBatch SQL语句（上创建语句对象）的方法。
- 执行使用executeBatch()将方法上创建表对象中的所有SQL语句。
- 最后，提交使用commit()方法的所有更改。

## 例如：

下面的代码段提供了使用Statement对象批量更新中的一个例子：

```
// Create statement object
Statement stmt = conn.createStatement();

// Set auto-commit to false
conn.setAutoCommit(false);

// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(200,'Zia', 'Ali', 30)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(201,'Raj', 'Kumar', 35)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "UPDATE Employees SET age = 35 " +
             "WHERE id = 100";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

为了更好地理解，建议学习研究JDBC批处理用Statement对象示例代码.

## 批处理使用prepareStatement结果对象：

下面是步骤，使用批处理用prepareStatement结果对象的典型顺序：

- 创建SQL语句的占位符。
- 使用任一prepareStatement()方法创建prepareStatement结果对象。
- 设置使用setAutoCommit()自动提交为false。
- 添加任意多个批量使用addBatch SQL语句（上创建语句对象）的方法。
- 执行使用executeBatch()将方法上创建表对象中的所有SQL语句。
- 最后，提交使用commit()方法的所有更改。

下面的代码段提供了使用prepareStatement结果对象批量更新的一个例子：

```
// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(?, ?, ?, ?)";

// Create PrepareStatement object
PreparedStatemen pstmt = conn.prepareStatement(SQL);

//Set auto-commit to false
conn.setAutoCommit(false);

// Set the variables
pstmt.setInt( 1, 400 );
pstmt.setString( 2, "Pappu" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 33 );
// Add it to the batch
pstmt.addBatch();

// Set the variables
pstmt.setInt( 1, 401 );
pstmt.setString( 2, "Pawan" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 31 );
// Add it to the batch
pstmt.addBatch();

//add more batches
.
.
.
.
//Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

为了更好地理解，建议学习研究实例代码.

# JDBC存储过程 - JDBC教程

在前而我们已经学习了如何使用使用JDBC存储过程，JDBC Statements。本教程是类似，但它会将有关JDBC的SQL转义语法的附加信息。

正如一个Connection对象创建Statement和PreparedStatement对象，它也创造了CallableStatement对象这将被用来执行调用数据库存储过程。

## 创建CallableStatement对象：

假设，需要执行以下Oracle存储过程：

```
CREATE OR REPLACE PROCEDURE getEmpName
   (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END;
```

注意: 上面已经写过Oracle存储过程，但我们正在使用MySQL数据库，写相同的存储过程对于MySQL如下，以EMP数据库中创建它：

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

三种类型的参数有：IN，OUT和INOUT。PreparedStatement对象只使用IN参数。CallableStatement对象可以使用所有的三个。

这里是每个定义：

| 参数 | 描述 |
| --- | --- |
| IN | 它的值是在创建SQL语句时未知的参数。将值绑定到setXXX（）方法的参数。 |
| OUT | 其值是由它返回的SQL语句提供的参数。你从OUT参数的getXXX（）方法检索值。 |
| INOUT | 同时提供输入和输出值的参数。绑定setXXX（）方法的变量，并与getXXX（）方法检索值。 |

下面的代码片段显示了如何使用该Connection.prepareCall（）方法实例化基于上述存储过程CallableStatement对象：

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

String变量的SQL表示存储过程，使用参数占位符。

使用CallableStatement对象是使用PreparedStatement对象。必须将值绑定到所有的参数执行该语句之前，否则将收到一个SQLException。

如果有IN参数，只要按照适用于PreparedStatement对象相同的规则和技巧;使用对应于要绑定Java数据类型的setXXX（）方法。

当使用OUT和INOUT参数就必须采用额外CallableStatement方法的registerOutParameter（）。registerOutParameter（）方法JDBC数据类型绑定到数据类型的存储过程返回。

一旦调用存储过程，用getXXX（）方法的输出参数检索值。这种方法投射SQL类型的值检索到Java数据类型。

# 关闭CallableStatement 对象：

正如关闭其他Statement对象，出于同样的原因，也应该关闭CallableStatement对象。

close（）方法简单的调用将完成这项工作。如果关闭了Connection对象首先它会关闭CallableStatement对象为好。然而，应该始终明确关闭的CallableStatement对象，以确保正确的清除。

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

为了更好地理解，建议研究学习Callable实例代码.

# JDBC的SQL转义语法：

转义语法使能够使用通过使用标准的JDBC方法和属性，无法使用数据库的某些特性的灵活性。

一般的SQL转义语法格式如下：

```
{keyword 'parameters'}
```

这里有以下这些，会发现非常有用的，而这样做的JDBC编程的转义序列：

# d, t, ts 关键字:

他们帮助确定日期，时间和时间戳记文字。如所知，没有两个数据库管理系统是基于时间和日期的方式相同。此转义语法告诉驱动程序呈现在目标数据库的格式，日期或时间。实现例子：

```
{d 'yyyy-mm-dd'}
```

其中yyyy=年，mm =月，DD =日。使用这种语法 {d '2009-09-03'}是2009年3月9日。

下面是一个简单的例子说明如何插入日期表：

```
//Create a Statement object
stmt = conn.createStatement();
//Insert data ==> ID, First Name, Last Name, DOB
String sql="INSERT INTO STUDENTS VALUES" +
             "(100,'Zara','Ali', {d '2001-12-16'})";

stmt.executeUpdate(sql);
```

同样，可以使用以下两种语法之一，无论是 t 或 ts：

```
{t 'hh:mm:ss'}
```

其中hh=小时，mm=分，ss=秒。使用此语法 {t '13:30:29'}是下午1点三十分29秒.

```
{ts 'yyyy-mm-dd hh:mm:ss'}
```

这是上述两种语法 'd' 和 't' 来表示时间戳结合语法。

# escape 关键字:

该关键字标识LIKE子句中使用的转义字符。有用使用SQL通配符%，其中匹配零个或多个字符时。例如：

```
String sql = "SELECT symbol FROM MathSymbols
              WHERE symbol LIKE '\%' {escape ''}";
stmt.execute(sql);
```

如果使用反斜杠字符（）作为转义字符，还必须使用两个反斜杠字符在Java字符串字面，因为反斜杠也是一个Java转义字符。

# fn 关键字:

此关键字代表在DBMS中使用标量函数。例如，可以使用SQL length函数计算GE字符串的长度：

```
{fn length('Hello World')}
```

这将返回11，字符串 'Hello World'的长度。.

# call 关键字:

此关键字是用来调用存储过程。例如，对于一个存储过程，需要一个IN参数，请使用以下语法：

```
{call my_procedure(?)};
```

对于一个存储过程，需要一个IN参数并返回一个OUT参数，使用下面的语法：

```
{? = call my_procedure(?)};
```

# oj 关键字:

此关键字用来表示外部联接。其语法如下：

```
{oj outer-join}
```

外连接表={LEFT| RIGHT| FULL}外连接{表|外连接}的搜索条件。例如：

```
String sql = "SELECT Employees
              FROM {oj ThisTable RIGHT
              OUTER JOIN ThatTable on id = '100'}";
stmt.execute(sql);
```

# JDBC流ASCII和二进制数据 - JDBC教程

PreparedStatement对象必须使用输入和输出流提供参数数据的能力。这使能够将整个文件到数据库列，可容纳较大的值，如CLOB和BLOB数据类型。

有下列方法，可用于将数据传送：

- setAsciiStream(): 这个方法是用来提供大的ASCII值。
- setCharacterStream(): 这个方法是用来提供大的UNICODE值。
- setBinaryStream(): 这个方法是用来提供大的二进制值。

setXXXStream()方法需要一个额外的参数，文件大小，除了参数占位符。这个参数通知驱动有多少数据要使用的流被发送到数据库中。

## 例子

考虑到要上传一个XML文件XML_Data.xml到数据库表。下面是这个XML文件的内容：

```
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
<Employee>
```

保持这个XML文件在要运行这个例子相同的目录。

这个例子将创建一个数据库表XML_Data，然后提交XML_Data.xml将被上传到该表中。

复制下面的例子JDBCExample.java，编译并运行，如下所示：

```
// Import required packages
import java.sql.*;
import java.io.*;
import java.util.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";
```

```
   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   PreparedStatement pstmt = null;
   Statement stmt = null;
   ResultSet rs = null;
   try{
      // Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      // Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //Create a Statement object and build table
      stmt = conn.createStatement();
      createXMLTable(stmt);

      //Open a FileInputStream
      File f = new File("XML_Data.xml");
      long fileLength = f.length();
      FileInputStream fis = new FileInputStream(f);

      //Create PreparedStatement and stream data
      String SQL = "INSERT INTO XML_Data VALUES (?,?)";
      pstmt = conn.prepareStatement(SQL);
      pstmt.setInt(1,100);
      pstmt.setAsciiStream(2,fis,(int)fileLength);
      pstmt.execute();

      //Close input stream
      fis.close();

      // Do a query to get the row
      SQL = "SELECT Data FROM XML_Data WHERE id=100";
      rs = stmt.executeQuery (SQL);
      // Get the first row
      if (rs.next ()){
         //Retrieve data from input stream
         InputStream xmlInputStream = rs.getAsciiStream (1);
         int c;
         ByteArrayOutputStream bos = new ByteArrayOutputStream();
         while (( c = xmlInputStream.read ()) != -1)
            bos.write(c);
         //Print results
         System.out.println(bos.toString());
      }
      // Clean-up environment
      rs.close();
      stmt.close();
```

```
      pstmt.close();
      conn.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(pstmt!=null)
            pstmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main

public static void createXMLTable(Statement stmt)
   throws SQLException{
   System.out.println("Creating XML_Data table..." );
   //Create SQL Statement
   String streamingDataSql = "CREATE TABLE XML_Data " +
                             "(id INTEGER, Data LONG)";
   //Drop table first if it exists.
   try{
      stmt.executeUpdate("DROP TABLE XML_Data");
   }catch(SQLException se){
   }// do nothing
   //Build table.
   stmt.executeUpdate(streamingDataSql);
}//end createXMLTable
}//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to database...
Creating XML_Data table...
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
<Employee>
Goodbye!
C:>
```

# JDBC Statement对象实例 - JDBC教程

以下是利用以下三种查询以及打开和关闭说明的例子：

- **boolean execute(String SQL) :** 返回一个布尔值true，如果ResultSet对象可以被检索，否则返回false。使用这个方法来执行SQL DDL语句，或当需要使用真正的动态SQL。
- **int executeUpdate(String SQL) :** 返回受影响的SQL语句执行的行数。使用此方法来执行，而希望得到一些受影响的行的SQL语句 - 例如，INSERT，UPDATE或DELETE语句。
- **ResultSet executeQuery(String SQL) :** 返回ResultSet对象。当希望得到一个结果集使用此方法，就像使用一个SELECT语句。

基于对环境和数据库安装在前面的章节中做此示例代码已被写入。

复制下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();
      String sql = "UPDATE Employees set age=30 WHERE id=103";

      // Let us check if it returns a true Result Set or not.
      Boolean ret = stmt.execute(sql);
```

```
      System.out.println("Return value is : " + ret.toString() );

      // Let us update age of the record with ID = 103;
      int rows = stmt.executeUpdate(sql);
      System.out.println("Rows impacted : " + rows );

      // Let us select all the records and display them.
      sql = "SELECT id, first, last, age FROM Employees";
      ResultSet rs = stmt.executeQuery(sql);

      //STEP 5: Extract data from result set
      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
      //STEP 6: Clean-up environment
      rs.close();
      stmt.close();
      conn.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to database...
Creating statement...
Return value is : false
Rows impacted : 1
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 30, First: Sumit, Last: Mittal
Goodbye!
C:>
```

# JDBC创建数据库实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序数据库的例子。执行下面的示例之前，请确保已到位如下：

- 应该有管理员特权，在给定的模式创建一个数据库。执行下面的例子中，需要与实际用户名和密码代替用户名和密码。
- MySQL或者其他数据库，正在使用的是启动和运行。

## 所需的步骤：

有创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入数据包 . 要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用导入的java.sql.* 就足够了。
- 注册**JDBC**驱动程序 . 要求初始化驱动程序，使可以与数据库打开一个通信通道。
- 打开连接 . 要求使用DriverManager.getConnection()方法创建一个Connection对象，它代表datbase服务器的物理连接。

  要创建一个新的数据库，不用给任何数据库名，同时准备数据库URL中所提到的下面的例子。
- 执行查询. 需要使用类型声明的对象建立并提交一个SQL语句到数据库。
- 清理环境**.** 需要明确地关闭所有的数据库资源与依靠JVM垃圾收集。

## 示例代码：

复制下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
```

```
    Connection conn = null;
    Statement stmt = null;
    try{
       //STEP 2: Register JDBC driver
       Class.forName("com.mysql.jdbc.Driver");

       //STEP 3: Open a connection
       System.out.println("Connecting to database...");
       conn = DriverManager.getConnection(DB_URL, USER, PASS);

       //STEP 4: Execute a query
       System.out.println("Creating database...");
       stmt = conn.createStatement();

       String sql = "CREATE DATABASE STUDENTS";
       stmt.executeUpdate(sql);
       System.out.println("Database created successfully...");
    }catch(SQLException se){
       //Handle errors for JDBC
       se.printStackTrace();
    }catch(Exception e){
       //Handle errors for Class.forName
       e.printStackTrace();
    }finally{
       //finally block used to close resources
       try{
          if(stmt!=null)
             stmt.close();
       }catch(SQLException se2){
       }// nothing we can do
       try{
          if(conn!=null)
             conn.close();
       }catch(SQLException se){
          se.printStackTrace();
       }//end finally try
    }//end try
    System.out.println("Goodbye!");
 }//end main
 }//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to database...
Creating database...
Database created successfully...
Goodbye!
C:>
```

# JDBC选择数据库实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序选择一个数据库的例子。执行下面的示例之前，请确保已做好如下工作：

- 执行下面的例子中，需要使用实际用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用(启动和运行)。

## 所需的步骤：

创建使用JDBC应用程序访问数据库需要执行以下步骤：

- 导入数据包 . 要求包括含有需要进行数据库编程的JDBC类包。大多数情况下，使用 import java.sql.*就足够了。
- 注册**JDBC**驱动程序 . 要求初始化驱动程序，从而可以与数据库打开一个通信通道。
- 打开连接. 使用DriverManager.getConnection()方法创建一个Connection对象，它代表使用所选（selected）数据库的物理连接。

  选择数据库时，准备数据库URL。下面的例子会让STUDENTS数据库连接。
- 清理环境. 需要明确地关闭所有的数据库资源相对依靠JVM的垃圾收集。

## 示例代码：

复制下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Goodbye!
C:>
```

# JDBC删除/Delete数据库实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序来删除一个现有数据库的例子。执行下面的示例之前，请确保如下：

- 执行下面的例子中，需要使用实际用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

注意：这是一个严重的操作，数据库中拥有的一切将会丢失。

## 所需步骤：

有创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入数据包：要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 就可以了.
- 注册**JDBC**驱动程序：要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开一个连接：要求中使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。

  删除一个数据库不需要数据库名称在数据库URL。下面的例子将删除STUDENTS数据库。
- 执行查询： 需要使用类型声明的对象建立并提交一个SQL语句来删除数据库。
- 清理环境. 需要明确地关闭所有的数据库资源相对依靠JVM的垃圾收集。

## 示例代码：

复制下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";
```

```
   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");

      //STEP 4: Execute a query
      System.out.println("Deleting database...");
      stmt = conn.createStatement();

      String sql = "DROP DATABASE STUDENTS";
      stmt.executeUpdate(sql);
      System.out.println("Database deleted successfully...");
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            conn.close();
      }catch(SQLException se){
      }// do nothing
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Deleting database...
Database deleted successfully...
Goodbye!
C:>
```

# JDBC创建表/Create实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序来创建表的例子。执行下面的示例之前，请确保已做好如下：

- 执行下面的例子中，你可以与你的实际的用户名和密码代替user name和password。
- MySQL或者其他数据库，正在使用: 启动和运行。

## 所需的步骤：

使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入数据包: 要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 就可以了.
- 注册**JDBC**驱动程序: 要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开连接: 要求中使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行查询: 需要使用类型声明的对象建立并提交一个SQL语句在入围数据库中创建表。
- 清理环境. 需要明确地关闭所有的数据库资源相对依靠JVM的垃圾收集。

## 示例代码：

复制下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
```

```
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");

      //STEP 4: Execute a query
      System.out.println("Creating table in given database...");
      stmt = conn.createStatement();

      String sql = "CREATE TABLE REGISTRATION " +
                   "(id INTEGER not NULL, " +
                   " first VARCHAR(255), " +
                   " last VARCHAR(255), " +
                   " age INTEGER, " +
                   " PRIMARY KEY ( id ))";

      stmt.executeUpdate(sql);
      System.out.println("Created table in given database...");
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            conn.close();
      }catch(SQLException se){
      }// do nothing
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating table in given database...
Created table in given database...
Goodbye!
C:>
```

# JDBC删除/Delete表实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序来删除一个表中的一个例子。执行下面的示例之前，请确保已经做好如下：

- 执行下面的例子中，使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

注: 这是一个严重的操作，在删除一个表前，要注意表中的数据备份，一旦删除表，表中的数据将消失。

## 所需的步骤：

创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入数据包：要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 就可以了.
- 注册**JDBC**驱动程序：要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开一个连接：要求使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行查询：需要使用类型声明的对象建立并提交一个SQL语句在所选择的数据库中删除表。
- 清理环境：需要明确地关闭所有的数据库资源相对依靠JVM的垃圾收集。

## 示例代码：

复制过去下面的例子JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
```

```
    Connection conn = null;
    Statement stmt = null;
    try{
       //STEP 2: Register JDBC driver
       Class.forName("com.mysql.jdbc.Driver");

       //STEP 3: Open a connection
       System.out.println("Connecting to a selected database...");
       conn = DriverManager.getConnection(DB_URL, USER, PASS);
       System.out.println("Connected database successfully...");

       //STEP 4: Execute a query
       System.out.println("Deleting table in given database...");
       stmt = conn.createStatement();

       String sql = "DROP TABLE REGISTRATION ";

       stmt.executeUpdate(sql);
       System.out.println("Table  deleted in given database...");
    }catch(SQLException se){
       //Handle errors for JDBC
       se.printStackTrace();
    }catch(Exception e){
       //Handle errors for Class.forName
       e.printStackTrace();
    }finally{
       //finally block used to close resources
       try{
          if(stmt!=null)
             conn.close();
       }catch(SQLException se){
       }// do nothing
       try{
          if(conn!=null)
             conn.close();
       }catch(SQLException se){
          se.printStackTrace();
       }//end finally try
    }//end try
    System.out.println("Goodbye!");
 }//end main
 }//end JDBCExample
```

现在编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Deleting table in given database...
Table  deleted in given database...
Goodbye!
C:>
```

# JDBC插入/Insert记录示例 - JDBC教程

本教程介绍了如何使用JDBC应用程序中插入表中的记录的例子。执行下面的示例之前，请确保如下：

- 执行下面的例子，需要使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

## 所需的步骤：

创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入数据包： 要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 就可以了。
- 注册**JDBC**驱动程序：要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开一个连接：要求中使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行查询：需要使用类型Statement对象建立并提交一个SQL语句将记录插入到表中。
- 清理环境：需要明确地关闭所有的数据库资源相对依靠JVM的垃圾收集。

## 示例代码：

复制过去下面的例子JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
```

```
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");

      //STEP 4: Execute a query
      System.out.println("Inserting records into the table...");
      stmt = conn.createStatement();

      String sql = "INSERT INTO Registration " +
                   "VALUES (100, 'Zara', 'Ali', 18)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration " +
                   "VALUES (101, 'Mahnaz', 'Fatma', 25)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration " +
                   "VALUES (102, 'Zaid', 'Khan', 30)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration " +
                   "VALUES(103, 'Sumit', 'Mittal', 28)";
      stmt.executeUpdate(sql);
      System.out.println("Inserted records into the table...");

   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            conn.close();
      }catch(SQLException se){
      }// do nothing
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Inserting records into the table...
Inserted records into the table...
Goodbye!
C:>
```

# JDBC查询Select记录实例 - JDBC教程

本教程介绍了如何选择/select/使用JDBC应用程序从一个表中获取记录的例子。执行下面的示例之前，请确保如下：

- 执行下面的例子中，使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

## 所需的步骤：

创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入数据包：要求您包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 就可以了。
- 注册**JDBC**驱动程序：要求初始化驱动程序，使它可以数据库打开一个通信通道。
- 打开一个连接：要求中使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行一个查询：需要使用Statement 类型对象，并提交一个SQL语句来选择（即取）从表中的记录。
- 提取数据：当SQL查询执行时，可以从表中提取记录。
- 清理环境：需要明确地关闭所有的数据库资源，不用依靠JVM的垃圾收集。

## 示例代码：

复制过去下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
```

```
    Statement stmt = null;
    try{
       //STEP 2: Register JDBC driver
       Class.forName("com.mysql.jdbc.Driver");

       //STEP 3: Open a connection
       System.out.println("Connecting to a selected database...");
       conn = DriverManager.getConnection(DB_URL, USER, PASS);
       System.out.println("Connected database successfully...");

       //STEP 4: Execute a query
       System.out.println("Creating statement...");
       stmt = conn.createStatement();

       String sql = "SELECT id, first, last, age FROM Registration";
       ResultSet rs = stmt.executeQuery(sql);
       //STEP 5: Extract data from result set
       while(rs.next()){
          //Retrieve by column name
          int id  = rs.getInt("id");
          int age = rs.getInt("age");
          String first = rs.getString("first");
          String last = rs.getString("last");

          //Display values
          System.out.print("ID: " + id);
          System.out.print(", Age: " + age);
          System.out.print(", First: " + first);
          System.out.println(", Last: " + last);
       }
       rs.close();
    }catch(SQLException se){
       //Handle errors for JDBC
       se.printStackTrace();
    }catch(Exception e){
       //Handle errors for Class.forName
       e.printStackTrace();
    }finally{
       //finally block used to close resources
       try{
          if(stmt!=null)
             conn.close();
       }catch(SQLException se){
       }// do nothing
       try{
          if(conn!=null)
             conn.close();
       }catch(SQLException se){
          se.printStackTrace();
       }//end finally try
    }//end try
    System.out.println("Goodbye!");
 }//end main
```

```
}//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:>
```

# JDBC更新/Update记录实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序更新表中的记录的一个例子。执行下面的示例之前，请确保如下：

- 执行下面的例子中，使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

## 所需的步骤：

创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入包：要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 *import java.sql.** 就可以了。
- 注册JDBC驱动程序：要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开一个连接：要求使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行一个查询：需要使用类型声明的对象建立并提交一个SQL语句来更新表中的记录。这个查询利用IN的和WHERE子句来更新条件的记录。
- 清理环境：需要明确地关闭所有的数据库资源相对于依靠JVM的垃圾收集。

## 示例代码：

复制过去下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
```

```
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();
      String sql = "UPDATE Registration " +
                   "SET age = 30 WHERE id in (100, 101)";
      stmt.executeUpdate(sql);

      // Now you can extract all the records
      // to see the updated records
      sql = "SELECT id, first, last, age FROM Registration";
      ResultSet rs = stmt.executeQuery(sql);

      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
      rs.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            conn.close();
      }catch(SQLException se){
      }// do nothing
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
```

```
    }//end try
    System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 101, Age: 30, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:>
```

# JDBC删除/Delete记录示例 - JDBC教程

本教程介绍了如何使用JDBC应用程序的表删除记录的例子。执行下面的示例之前，请确保以下：

- 执行下面的例子中，使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

## 所需的步骤：

创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入包：要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 就可以了。
- 注册**JDBC**驱动程序：要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开一个连接：要求使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行一个查询：需要使用类型声明的对象建立并提交一个SQL语句从表中删除记录。这使得查询使用WHERE子句来删除条件的记录。
- 清理环境：需要明确地关闭所有的数据库资源相对于依靠JVM的垃圾收集。

## 示例代码：

复制过去下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
```

```
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();
      String sql = "DELETE FROM Registration " +
                   "WHERE id = 101";
      stmt.executeUpdate(sql);

      // Now you can extract all the records
      // to see the remaining records
      sql = "SELECT id, first, last, age FROM Registration";
      ResultSet rs = stmt.executeQuery(sql);

      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
      rs.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            conn.close();
      }catch(SQLException se){
      }// do nothing
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
```

```
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:>
```

# JDBC WHERE子句实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序表中选择记录的一个例子。这将使用WHERE子句，而从表中选择记录添加附加条件。执行下面的示例之前，请确保如下：

- 执行下面的例子中，使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

## 所需的步骤：

创建使用JDBC应用程序一个新的数据库需要执行以下步骤：

- 导入包：要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 。
- 注册**JDBC**驱动程序：要求初始化驱动程序，使可以与数据库打开一个通信通道。
- 打开一个连接：要求使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行一个查询：需要使用类型声明的对象建立并提交一个SQL语句从符合给定条件的表获取记录。查询使用WHERE子句来查询记录。
- 清理环境：需要明确地关闭所有的数据库资源相对于依靠JVM的垃圾收集。

## 示例代码：

复制过去的下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
```

```
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();

      // Extract records without any condition.
      System.out.println("Fetching records without condition...");
      String sql = "SELECT id, first, last, age FROM Registration";
      ResultSet rs = stmt.executeQuery(sql);

      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }

      // Select all records having ID equal or greater than 101
      System.out.println("Fetching records with condition...");
      sql = "SELECT id, first, last, age FROM Registration" +
                   " WHERE id >= 101 ";
      rs = stmt.executeQuery(sql);

      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
      rs.close();
   }catch(SQLException se){
      //Handle errors for JDBC
```

```
         se.printStackTrace();
      }catch(Exception e){
         //Handle errors for Class.forName
         e.printStackTrace();
      }finally{
         //finally block used to close resources
         try{
            if(stmt!=null)
               conn.close();
         }catch(SQLException se){
         }// do nothing
         try{
            if(conn!=null)
               conn.close();
         }catch(SQLException se){
            se.printStackTrace();
         }//end finally try
      }//end try
      System.out.println("Goodbye!");
   }//end main
   }//end JDBCExample
```

现在让我们来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
Fetching records without condition...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Fetching records with condition...
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:>
```

# JDBC LIKE子句实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序表中选择记录的一个例子。这里使用LIKE子句，而从表中选择记录添加附加条件。执行下面的示例之前，请确保如下：

- 执行下面的例子中，请使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

## 所需步骤：

使用JDBC应用程序创建一个新的数据库需要执行以下步骤：

- 导入包：要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 。
- 注册JDBC驱动程序：要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开一个连接：要求使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行一个查询：需要使用类型Statement对象建立并提交一个SQL语句从符合给定条件的表获取记录。这使得查询使用LIKE子句来选择记录选择所有名字以 "za" 开始的学生。
- 清理环境：需要明确地关闭所有的数据库资源相对于依靠JVM的垃圾收集。

## 示例代码：

复制过去的下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
```

```
    Statement stmt = null;
    try{
       //STEP 2: Register JDBC driver
       Class.forName("com.mysql.jdbc.Driver");

       //STEP 3: Open a connection
       System.out.println("Connecting to a selected database...");
       conn = DriverManager.getConnection(DB_URL, USER, PASS);
       System.out.println("Connected database successfully...");

       //STEP 4: Execute a query
       System.out.println("Creating statement...");
       stmt = conn.createStatement();

       // Extract records without any condition.
       System.out.println("Fetching records without condition...");
       String sql = "SELECT id, first, last, age FROM Registration";
       ResultSet rs = stmt.executeQuery(sql);

       while(rs.next()){
          //Retrieve by column name
          int id  = rs.getInt("id");
          int age = rs.getInt("age");
          String first = rs.getString("first");
          String last = rs.getString("last");

          //Display values
          System.out.print("ID: " + id);
          System.out.print(", Age: " + age);
          System.out.print(", First: " + first);
          System.out.println(", Last: " + last);
       }

       // Select all records having ID equal or greater than 101
       System.out.println("Fetching records with condition...");
       sql = "SELECT id, first, last, age FROM Registration" +
                    " WHERE first LIKE '%za%' ";
       rs = stmt.executeQuery(sql);

       while(rs.next()){
          //Retrieve by column name
          int id  = rs.getInt("id");
          int age = rs.getInt("age");
          String first = rs.getString("first");
          String last = rs.getString("last");

          //Display values
          System.out.print("ID: " + id);
          System.out.print(", Age: " + age);
          System.out.print(", First: " + first);
          System.out.println(", Last: " + last);
       }
       rs.close();
```

```
    }catch(SQLException se){
       //Handle errors for JDBC
       se.printStackTrace();
    }catch(Exception e){
       //Handle errors for Class.forName
       e.printStackTrace();
    }finally{
       //finally block used to close resources
       try{
          if(stmt!=null)
             conn.close();
       }catch(SQLException se){
       }// do nothing
       try{
          if(conn!=null)
             conn.close();
       }catch(SQLException se){
          se.printStackTrace();
       }//end finally try
    }//end try
    System.out.println("Goodbye!");
 }//end main
 }//end JDBCExample
```

现在来编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
Fetching records without condition...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Fetching records with condition...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
Goodbye!
C:>
```

# JDBC排序实例 - JDBC教程

本教程介绍了如何使用JDBC应用程序的对表中的数据进行排序的例子。这将使用asc和desc关键字的记录按升序或降序排序。执行下面的示例之前，请确保如下：

- 执行下面的例子中，请使用实际的用户名和密码代替username和password。
- MySQL或者其他数据库，正在使用：启动和运行。

## 所需步骤：

使用JDBC应用程序创建一个新的数据库需要执行以下步骤：

- 导入包：要求包括含有需要进行数据库编程的JDBC类的包。大多数情况下，使用 import java.sql.* 。
- 注册**JDBC**驱动程序：要求初始化驱动程序，使它可以与数据库打开一个通信通道。
- 打开一个连接：要求使用DriverManager.getConnection()方法创建一个Connection对象，它代表与数据库服务器的物理连接。
- 执行一个查询：需要使用类型语句的对象建立并提交一个SQL语句的记录从表中进行排序。查询利用ASC和DESC子句来按升序和降序顺序排序数据。
- 清理环境：需要明确地关闭所有的数据库资源相对于依靠JVM的垃圾收集。

## 示例代码：

复制过去的下面的例子中JDBCExample.java，编译并运行，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
```

```
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to a selected database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      System.out.println("Connected database successfully...");

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();

      // Extract records in ascending order by first name.
      System.out.println("Fetching records in ascending order...");
      String sql = "SELECT id, first, last, age FROM Registration" +
                   " ORDER BY first ASC";
      ResultSet rs = stmt.executeQuery(sql);

      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }

      // Extract records in descending order by first name.
      System.out.println("Fetching records in descending order...");
      sql = "SELECT id, first, last, age FROM Registration" +
                   " ORDER BY first DESC";
      rs = stmt.executeQuery(sql);

      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
      rs.close();
   }catch(SQLException se){
```

```
         //Handle errors for JDBC
         se.printStackTrace();
      }catch(Exception e){
         //Handle errors for Class.forName
         e.printStackTrace();
      }finally{
         //finally block used to close resources
         try{
            if(stmt!=null)
               conn.close();
         }catch(SQLException se){
         }// do nothing
         try{
            if(conn!=null)
               conn.close();
         }catch(SQLException se){
            se.printStackTrace();
         }//end finally try
      }//end try
      System.out.println("Goodbye!");
   }//end main
   }//end JDBCExample
```

现在编译上面的例子如下：

```
C:>javac JDBCExample.java
C:>
```

当运行JDBCExample，它会产生以下结果：

```
C:>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
Fetching records in ascending order...
ID: 103, Age: 28, First: Sumit, Last: Mittal
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 100, Age: 30, First: Zara, Last: Ali
Fetching records in descending order...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:>
```

# JDBC快速入门教程 - JDBC教程

# JDBC是什么?

JDBC API是一个Java API，可以访问任何类型表列数据，特别是存储在关系数据库中的数据。JDBC代表Java数据库连接。

JDBC库中所包含的API任务通常与数据库使用：

- 连接到数据库
- 创建SQL或MySQL语句
- 在数据库中执行SQL或MySQL查询
- 查看和修改记录

123

# 先决条件:

学习JDBC，需要在以下两个主题有一定的了解：

1. JAVA核心编程
2. SQL或MySQL数据库

# JDBC - 环境设置:

请确认您已完成以下设置：

1. 核心JAVA安装
2. SQL 或 MySQL数据库安装

除上述者外，需要建立一个数据库，为本程测试项目使用。假设这是EMP，在同一个数据库上创建表Employees。

# 创建JDBC应用程序:

参与建立一个JDBC应用程序，本教程中按六个步骤进行：

## 导入包:

这需要你有软件包包含了数据库编程所需的JDBC类。大多数情况下，使用import java.sql.* 就足够了，如下所示：

```
//STEP 1\. Import required packages
import java.sql.*;
```

## 注册JDBC驱动程序：

这需要初始化驱动程序，这样就可以打开与数据库的通信信道。以下是代码片段实现这一目标：

```
//STEP 2: Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");
```

## 打开一个连接：

这需要使用DriverManager.getConnection()方法来创建一个Connection对象，它代表一个物理连接的数据库，如下所示：

```
//STEP 3: Open a connection
//  Database credentials
static final String USER = "username";
static final String PASS = "password";
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

## 执行一个查询：

这需要使用一个对象类型Statement或PreparedStatement构建，并提交一个SQL语句到数据库。如下：

```
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
```

如果有一个SQL UPDATE，INSERT或DELETE语句，那么需要下面的代码片段：

```
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "DELETE FROM Employees";
ResultSet rs = stmt.executeUpdate(sql);
```

## 从结果集中提取数据：

这一步是必需的情况下，从数据库中获取数据。可以使用适当的ResultSet.getXXX()方法来检索的数据结果如下：

```
//STEP 5: Extract data from result set
while(rs.next()){
    //Retrieve by column name
    int id  = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
}
```

## 清理环境：

应该明确地关闭所有的数据库资源，对依赖于JVM的垃圾收集如下：

```
//STEP 6: Clean-up environment
rs.close();
stmt.close();
conn.close();
```

# 第一个JDBC 程序:

基于上面的步骤，我们可以有以下综合示例代码，我们可以用它作为模板而写JDBC代码： 此示例代码已被写入基于对环境和数据库环境一章中设置完成。

```
//STEP 1\. Import required packages
import java.sql.*;

public class FirstExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();
      String sql;
      sql = "SELECT id, first, last, age FROM Employees";
      ResultSet rs = stmt.executeQuery(sql);

      //STEP 5: Extract data from result set
      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
```

```
      //STEP 6: Clean-up environment
      rs.close();
      stmt.close();
      conn.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end FirstExample - by www.yiibai.com
```

现在让我们来编译上面的例子如下：

```
C:\>javac FirstExample.java
C:\>
```

当你运行FirstExample的，它会产生以下结果：

```
C:\>java FirstExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
C:\>
```

# SQLException方法：

一个SQLException既可以发生在驱动程序和数据库。当这样的异常时，SQLException类型的对象将被传递到catch子句。 通过SQLException对象有以下几种方法可用于获取更多的关于异常的信息：

| 方法 | 描述 |
| --- | --- |
| getErrorCode() | 获取与异常关联的错误号。 |
| getMessage() | 获取的JDBC驱动程序的错误处理错误消息的驱动程序，或获取Oracle错误号和消息的数据库错误。 |
| getSQLState() | 获取XOPEN SQLSTATE字符串。对于JDBC驱动程序错误，没有有用的信息，从该方法返回。对于一个数据库错误，五位的XOPEN SQLSTATE代码返回。这种方法可以返回null。 |
| getNextException() | 获取异常链中的下一个Exception对象。 |
| printStackTrace() | 打印当前的异常，或可抛出，并回溯到标准错误流。 |
| printStackTrace(PrintStream s) | 打印此抛出对象及其回溯到指定的打印流。 |
| printStackTrace(PrintWriter w) | 打印此抛出对象及其回溯您指定打印作家。 |

通过利用从Exception对象提供的信息，可以捕获一个异常，并适当地继续运行程序。这是一个try块中的一般形式：

```
try {
   // Your risky code goes between these curly braces!!!
}
catch(Exception ex) {
   // Your exception handling code goes between these
   // curly braces, similar to the exception clause
   // in a PL/SQL block.
}
finally {
   // Your must-always-be-executed code goes between these
   // curly braces. Like closing database connection.
}
```

# JDBC - 数据类型:

下表总结了默认的JDBC数据类型的Java数据类型转换，当调用PreparedStatement或CallableStatement对象的setXXX()方法，或ResultSet.updateXXX()方法。

| SQL | JDBC/Java | setXXX | updateXXX |
|---|---|---|---|
| VARCHAR | java.lang.String | setString | updateString |
| CHAR | java.lang.String | setString | updateString |
| LONGVARCHAR | java.lang.String | setString | updateString |
| BIT | boolean | setBoolean | updateBoolean |
| NUMERIC | java.math.BigDecimal | setBigDecimal | updateBigDecim |
| TINYINT | byte | setByte | updateByte |
| SMALLINT | short | setShort | updateShort |
| INTEGER | int | setInt | updateInt |
| BIGINT | long | setLong | updateLong |
| REAL | float | setFloat | updateFloat |
| FLOAT | float | setFloat | updateFloat |
| DOUBLE | double | setDouble | updateDouble |
| VARBINARY | byte[ ] | setBytes | updateBytes |
| BINARY | byte[ ] | setBytes | updateBytes |
| DATE | java.sql.Date | setDate | updateDate |
| TIME | java.sql.Time | setTime | updateTime |
| TIMESTAMP | java.sql.Timestamp | setTimestamp | updateTimestam |
| CLOB | java.sql.Clob | setClob | updateClob |
| BLOB | java.sql.Blob | setBlob | updateBlob |
| ARRAY | java.sql.Array | setARRAY | updateARRAY |
| REF | java.sql.Ref | SetRef | updateRef |
| STRUCT | java.sql.Struct | SetStruct | updateStruct |

JDBC3.0的增强支持BLOB，CLOB，ARRAY，REF数据类型。的ResultSet对象UPDATEBLOB()，updateCLOB()，updateArray()和updateRef()方法，使您可以在服务器上直接操作相应的数据。

setXXX()和updateXXX()方法，可以转换成特定的Java类型到特定的JDBC数据类型。setObject()和updateObject()方法，几乎所有的Java类型映射到JDBC数据类型。

ResultSet对象提供相应的getXXX()方法为每个数据类型来检索列值。每一种方法，可以使用与列名或由它的序号位置。

| SQL | JDBC/Java | setXXX | getXXX |
|---|---|---|---|
| VARCHAR | java.lang.String | setString | getString |
| CHAR | java.lang.String | setString | getString |
| LONGVARCHAR | java.lang.String | setString | getString |
| BIT | boolean | setBoolean | getBoolean |
| NUMERIC | java.math.BigDecimal | setBigDecimal | getBigDecimal |
| TINYINT | byte | setByte | getByte |
| SMALLINT | short | setShort | getShort |
| INTEGER | int | setInt | getInt |
| BIGINT | long | setLong | getLong |
| REAL | float | setFloat | getFloat |
| FLOAT | float | setFloat | getFloat |
| DOUBLE | double | setDouble | getDouble |
| VARBINARY | byte[ ] | setBytes | getBytes |
| BINARY | byte[ ] | setBytes | getBytes |
| DATE | java.sql.Date | setDate | getDate |
| TIME | java.sql.Time | setTime | getTime |
| TIMESTAMP | java.sql.Timestamp | setTimestamp | getTimestamp |
| CLOB | java.sql.Clob | setClob | getClob |
| BLOB | java.sql.Blob | setBlob | getBlob |
| ARRAY | java.sql.Array | setARRAY | getARRAY |
| REF | java.sql.Ref | SetRef | getRef |
| STRUCT | java.sql.Struct | SetStruct | getStruct |

# JDBC - 批量处理:

批处理允许一个批处理组相关的SQL语句，并将其提交的一个调用到数据库。

当几个SQL语句一次发送到数据库中，可以减少通信开销，从而提高性能。

- JDBC驱动程序不支持此功能。您应该使用DatabaseMetaData.supportsBatchUpdates()方法来确定目标数据库支持批量更新处理。如果你的JDBC驱动程序支持此功能，则该方法返回true。
- addBatch()方法的声明，PreparedStatement和CallableStatementis用于添加单个语句的批处理。 executeBatch()将开始执行的所有语句组合到一起。
- executeBatch()将返回一个整数数组，每个数组元素的表示为相应的更新语句的更新计数。
- 可以添加语句进行批处理，可以clearBatch()方法删除它们。此方法将删除addBatch()方法添加的所有语句。但是，你不能有选择性地选择语句来删除。

# JDBC - 数据流:

PreparedStatement对象有能力使用提供参数数据的输入和输出流。这使您可以将整个文件到数据库中，可容纳较大的值，如CLOB和BLOB数据类型的列。

有下列方法可用于流数据：

1. setAsciiStream(): 此方法用于提供大的ASCII值。
2. setCharacterStream(): 此方法用于提供大的UNICODE值。
3. setBinaryStream(): 使用此方法，以提供大的二进制值。

setXXXStream()方法需要一个额外的参数，文件大小，除了参数占位符。此参数通知应发送多少数据的数据库，使用流的驱动程序。

对于一个详细的关于所有这些概念，需要去通过学习完整的教程。

# JFreeChart教程

图表是信息的图形表示。有可用的各种工具，它可用于创建不同类型的图表。

本教程学习什么是JFreeChart？为什么需要它，并在各种方式列出一个基于Java的应用程序或独立创建不同类型的图表。

## JFreeChart是什么?

JfreeChart是用Java开发的开源库，它可以在基于Java的应用程序可用于创建各种各样的图表。通过使用JFreeChart，可以创建2D和3D图表，如饼图，条形图，折线图，XY图和3D图表所有常用的主要类型。

## 为什么要使用JFreeChart?

- 它是100%开源和免费的，允许使用在商业应用中无需任何费用。
- 它配备了有据可查的API，这使得它很容易理解。
- 它支持多种图表类型，如饼图，折线图，条形图，面积图和三维图表。
- JFreeChart易于扩展，并且可以在客户端以及服务器端应用程序中被使用。
- 它支持多种输出格式，如PNG，JPEG，PDF，SVG等。
- 它允许图表丰富的自定义。

## 在哪里使用JFreeChart？

考虑有这样一个情况，当正在开发一个应用程序，需要以图表的形式显示数据，其中数据本身是动态填充。在这种情况下，可以使用JFreeChart显示，使用简单的编程图表的形式的数据。

## 历史

JFreeChart 工程由David Gilbert成立十四年前（2000年2月），而现在JFreeChart是在Java开发的最广泛使用的图表库。

# JFreeChart安装 - JFreeChart教程

本章将指导您完成JFreeChart在Windows和Linux的安装设置过程。所需的用户管理，同时安装JFreeChart。JFreeChart是著名的高效图表创建和用户友好的安装设置。

## 系统要求：

| JDK | 1.5 或以上 |
| --- | --- |
| 内存 | 2GB RAM |
| --- | --- |
| 硬盘空间 | 没有最低要求 |
| --- | --- |
| 操作系统版本 | Linux / Windows |
| --- | --- |

## 安装JFreeChart

要安装JFreeChart，首先需要在系统上安装Java。

### 步骤1：验证Java安装

为了验证Java安装，打开控制台并执行下面的Java命令：

| 操作系统 | 任务 | 命令 |
| --- | --- | --- |
| Windows | 打开命令控制台 | C:>java -version |
| Linux | 打开命令终端 | $java -version |

如果Java安装得当，那么应该得到两个操作系统下面的输出：

| 操作系统 | 描述 |
|---|---|
| Windows | Java version "1.7.0_60" Java (TM) SE Run Time Environment (build 1.7.0_60-b19) Java HotSpot(TM) 64-bit Server VM (build 24.60-b09,mixed mode) |
| Linux | java version "1.7.0_25" OpenJDK Runtime Environment (rhel-2.3.10.4.el6_4-x86_64) OpenJDK 64-Bit Server VM (build 23.7-b01, mixed mode) |

如果没有安装Java，那么可以链接到安装Java软件开发工具包（SDK）：

http://www.oracle.com/technetwork/java/javase/downloads/index.html

我们假设你已经进行了本教程之前安装了Java1.7.0_60 版本。

## 第2步：设置Java环境

设置 JAVA_HOME 环境变量指向到安装在机器上的 Java 基本目录的位置。例如，

| Os | 描述 |
|---|---|
| Windows | 设置的环境变量 JAVA_HOME 的值为 C:\ProgramFiles\java\jdk1.7.0_60 |
| Linux | export JAVA_HOME=/usr/local/java-current |

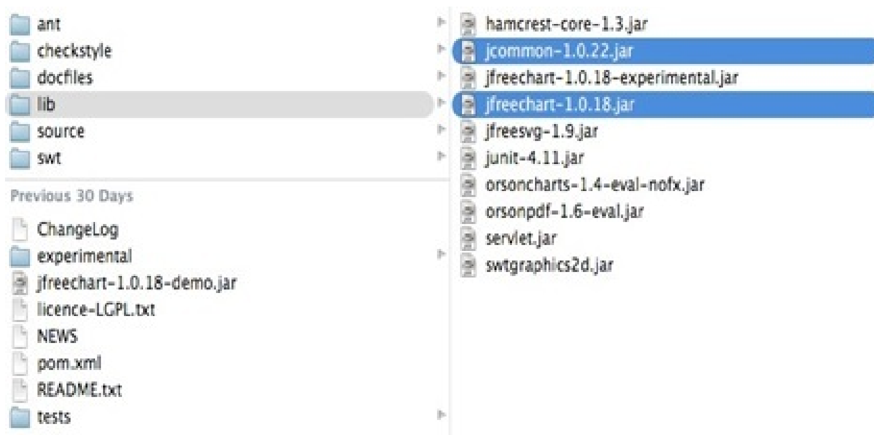
附加 Java编译器位置系统路径。

| OS | 描述 |
|---|---|
| Windows | 添加字符串；C:\Program Files\Java\jdk1.7.0_60\bin 到系统环境变量 PATH. |
| Linux | export PATH=$PATH:$JAVA_HOME/bin/ |

从命令提示符下验证命令 java-version 如上所述

## 第三步：安装 JFreeChart

从链接 http://www.jfree.org/jfreechart/download/ 下载JFreeChart.zip最新版本 解压下载文件库，可以链接到Java程序中的任何位置。下图显示了目录和文件的结构：




添加了JFreeChart1.0.18.jar 和 jcommon-1.0.22.jar 文件到 CLASSPATH 完整路径，如下图所示：

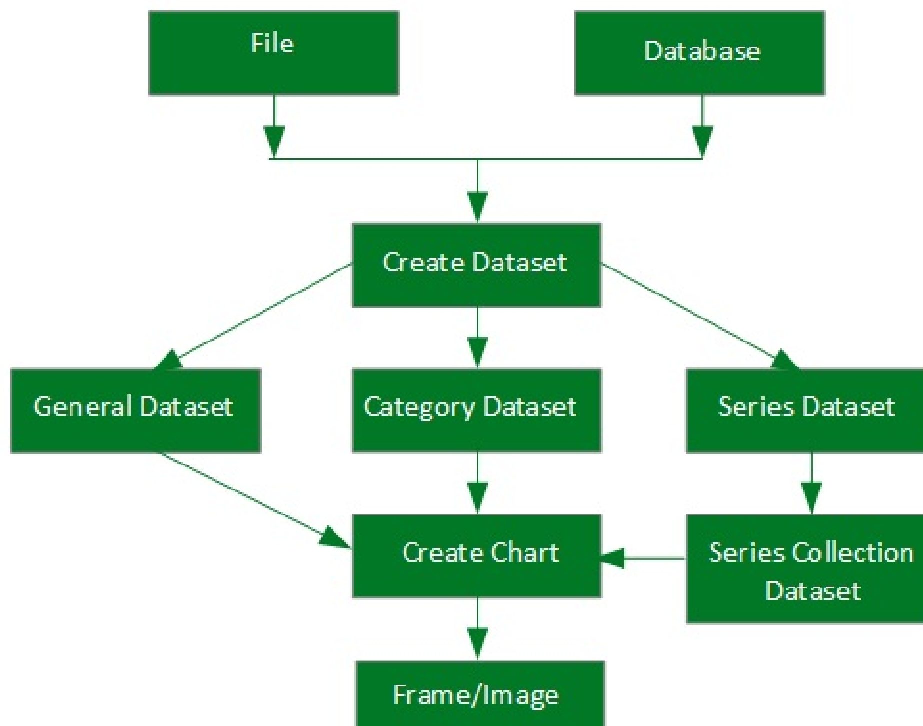
| OS | 描述 |
| --- | --- |
| Windows | 添加字符串 “C:\ jfreechart-1.0.18\lib\ jfreechart-1.0.18.jar” 和 “C:\ jfreechart-1.0.18\lib\ jcommon-1.0.22.jar” 到用户变量 CLASSPATH 的尾部 |
| Linux | Export CLASSPATH=$CLASSPATH: /usr/share/jfreechart-1.0.18/lib/jfreechart-1.0.18.jar: /usr/share/jfreechart-1.0.18/lib/jcommon-1.0.22.jar |

# JFreeChart架构 - JFreeChart教程

本章介绍给大家介绍 JFreeChart 不同类中如何交互的概念, JFreeChart基本类层次和应用水平的架构在基于Java应用程序如何工作的。

## 类层次架构

类层次架构解释了如何把不同阶层的相互库交互，以创建不同类型的图表。

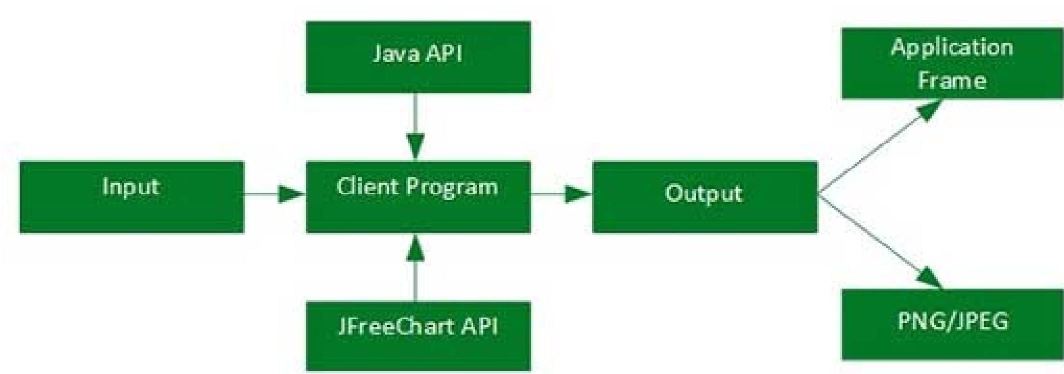



以下是在上述框图中使用的单元细节：

| 单元 | 描述 |
|---|---|
| 文件 | 所用的用户输入为源，用于创建该文件中的数据集。 |
| 数据库 | 所用的用户输入为源，用于创建在数据库中的数据集。 |
| 创建数据集 | 接受数据集中存储和数据集中到数据集对象。 |
| 通用数据集 | 这种类型的数据集主要用于饼图。 |
| 分类数据集 | 这种类型的数据集，用于柱状图，折线图等等。 |
| 系列数据集 | 这种类型的数据集被用于存储一系列数据和构建线图表。 |
| 系列采集数据集 | 不同类别的一系列数据集添加系列集合数据集。这种类型的数据集，用于xy折线图表。 |
| 创建图表 | 这是被执行以创建最终的图表的方法。 |
| 帧/图片 | 该图显示在一个Swing框架或创建映像。 |

## 应用层架构

应用级架构说明，其中JFreeChart库在Java应用程序内线。




客户端程序接收用户数据，然后它使根据要求使用标准Java和JFreeChart的API来生成输出在任一帧的形式，它可以直接在该应用程序或独立地在所述图像格式，如JPEG或PNG显示。

# JFreeChart参考API - JFreeChart教程

在本章中，我们将讨论一些在JFreeChart库重要的软件包，类和方法。这些软件包，类和方法是最常见的，同时建立了各种使用JFreeChart库图表。

## ChartFactory 类

ChartFactory是org.jfree.chart包中抽象类。它提供了实用方法的集合，用于生成标准的图表。以下是几个重要方法的列表：

### 类构造方法

| S.N. | 描述 |
|---|---|
| 1 | **ChartFactory()** ChartFactory类的默认构造函数。 |

### 类方法

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **createPieChart(java.lang.String title, PieDataset dataset, boolean legend, boolean tooltips, boolean urls)** 此方法使用默认设置创建一个饼图。它返回JFreeChart类型的对象。 |
| 2 | **createPieChart3D(java.lang.String title, PieDataset dataset, boolean legend, boolean tooltips, boolean urls)** 此方法使用指定的数据集三维/3D饼图。 |
| 3 | **createBarChart(java.lang.String title, java.lang.String categoryAxisLabel, java.lang.String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)** 参数java.lang.String categoryAxisLabel标签放置在X轴的值。该参数的java.lang.String valueAxisLabel标签放置在Y轴的数值。此方法创建一个条形图。 |
| 4 | **createBarChart3D(java.lang.String title, java.lang.String categoryAxisLabel, java.lang.String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)** 此方法创建一个柱形图具有3D效果。它返回JFreeChart类型的对象。 |
| 5 | **createLineChart(java.lang.String title, java.lang.String categoryAxisLabel, java.lang.String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)** 此方法使用默认设置创建一个折线图。 |
| 6 | **createLineChart3D(java.lang.String title, java.lang.String categoryAxisLabel, java.lang.String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)** 此方法创建一个折线图与3D效果。 |
| 7 | **createXYLineChart(java.lang.String title, java.lang.String xAxisLabel, java.lang.String yAxisLabel, XYDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)** 此方法使用默认设置创建基于XYDataset的折线图。 |

## ChartFrame 类

ChartFrame类在org.jfree.chart包中，提供所有的帧相关的功能和工具。ChartFrame类继承自父类，如Frame, Window, Container, Component 类功能。

### 类构造方法

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **ChartFrame** (java.lang.Frame String, JfreeChart chart) 它构建一个框架/frame。 |
| 2 | **Chart Frame (java.lang.Frame String, JfreeChart chart, boolean scrollpane)** 它构建一个框架/frame。 |

### 类方法

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **getChartPanel()** 此方法返回图表面板的框架/frame。 |

## ChartPanel 类

org.jfree.chart包中的ChartPanel类用于swing GUI部件，用于显示JfreeChart对象。

### 类构造方法

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **ChartPanel(JFreeChart chart)** 此构造一个面板/panel， 显示指定的图表。 |
| 2 | **ChartPanel(JFreeChart chart, boolean useBuffer)** 这个构造函数构造包含图表的面板/panel。 |
| 3 | **ChartPanel(JFreeChart chart, boolean properties, boolean save, boolean print, boolean zoom, boolean tooltips)** 此构造一个JFreeChart面板。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **setPreferredSize(java.awt.Dimension)** 该方法用于java.awt中设置的帧大小。 Dimension类对象作为参数。这个方法是从javax.swing.JComponent中实现。 |

## ChartUtilities 类

org.jfree.chart包中的CharUtilites类提供JFreeCharts包括将图表转换成图像文件格式，如PNG，JPEG和创建HTML图像映射方法的实用方法的集合。

### 类构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **ChartUtilities()** 这是类的一个默认构造函数 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **saveChartAsPNG(java.io.File file, JfreeChart chart, int width, int height)** 此方法转换和保存图表为PNG格式指定的文件。 |
| 2 | **saveChartAsJPEG(java.io.File file, JfreeChart chart, int width, int height)** 此方法转换并保存一个图表，以JPEG格式指定的文件。 |

## JFreeChart 类

JFreeChart 类是在org.jfree.chart包的核心类。这个类提供了JFreeChart的方法来创建柱状图，折线图，饼图和XY坐标图，包括时间序列数据。

### 类构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **JfreeChart(Plot plot)** 此构造函数创建基于所提供的节点一个新的图表。 |
| 2 | **JfreeChart(java.lang.String title, java.awt.Font titleFont, Plot plot, boolean createLegend)** 该构造函数创建一个新的图表给定标题和绘图。 |
| 3 | **JfreeChart(java.lang.String title, Plot plot)** 该构造函数创建一个新的图表给定标题和绘图。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **getXYPlot()** 此方法返回节点图表作为XYPlot。使用XYPolt我们可以XY图表做了一些实用操作。 |

# PiePlot 类

这个类是org.jfree.chart.plot包的一部分，来自同一个包扩展Plot 类。这个类提供了一些方法来创建饼图块。

## 类的构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **PiePlot()** 它创建了一个新的绘图。 |
| 2 | **PiePlot(PieDataset dataset)** 它创建了一个绘图，饼图由指定的数据集。 |

## 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **setStartAngle(double angle)** 此方法设置起始角度和发送PlotChangeEvent向所有注册的侦听器 |

# PiePlot3D 类

PiePlot3D类和PiePlot类在同一个包中的子类。因此，这两个类有相同的功能，PiePlot类只不过是用于创建3D图形。

## 类的构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **PiePlot3D()** 此构造函数创建没有数据集的新实例。 |
| 2 | **PiePlot3D(PieDataset dataset)** 该构造函数创建一个饼图和使用指定的数据集三维效果。 |

## 类方法

| S.N. | 方法及描述 |
| --- | --- |
| 1 | **setForegroundAlpha(float alpha)** 它设置alpha透明度并向所有发送PlotChangeEvent注册的侦听器。 |
| 2 | **setInteriorGap(double percent)** 它设置了内部差距并发送PlotChangeEvent向所有注册的侦听器。这种控制的饼图绘图边缘与绘图区本身（即，其中部分标签显示的区域）之间的空间。这个方法是从父类PiePlot继承。 |

# PlotOrientation 类

这是一个串行化类从org.jfree.chart.plot封装，它是用来显示一个二维曲线图的方位。方向可以是垂直的或水平的。它设置Y轴的方向。传统的绘图有一个垂直的Y轴。

## 字段摘要

| S.N. | 类型 | 字段 & 描述 |
| --- | --- | --- |
| 1 | PlotOrientation | **HORIZONTAL** 为一个曲线图，其中所述范围轴（Y轴）是水平。 |
| 2 | PlotOrientation | **VERTICAL** 为一个曲线图，其中所述范围轴（Y轴）是垂直的。这个默认的方向。 |

## 类方法

| S.N. | 方法及描述 |
| --- | --- |
| 1 | **isHorizontal()** 如果这个方向是水平的，此方法返回true，否则返回false。 |
| 2 | **isVertical()** 如果这个方向是VERTICAL，此方法返回true，否则返回false。 |

# XYPlot 类

这在org.jfree.chart.plot包可用一个通用类，并将其用于在（X，Y）对的形式标绘数据。这个曲线图可以从实现XYDataSet接口的任何其它类中使用的数据。 XYPlot利用一个XYItemRenderer的画在图上的每个点。

## 类的构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **XYPlot()** 该构造器没有数据集，无轴，无渲染器创建一个新的XYPlot实例。 |
| 2 | **XYPlot(XYDataset** dataset， ValueAxis domainAxis， ValueAxis rangeAxis， XYItemRenderer renderer），此构造函数创建一个新的绘图并指定数据集，轴和渲染。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **setRenderer(XYItemRenderer** renderer) 此方法设置渲染器的主要数据集，并发送更改事件向所有注册的侦听器。 |

## NumberAxis 类

这个类是org.jfree.chart.axis封装，它可以访问任意轴的数值数据。当我们设置任何轴的范围为默认值，它根据所述数据的范围配合。但使用NumberAxis，类我们可以设置较低的利润率和定义域和值域轴的上侧边距。

### 类的构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **NumberAxis( )** 这是NumberAxis一个默认的构造函数。 |
| 2 | **NumberAxis( java.lang.String label)** 构造函数NumberAxis使用必要的默认值在哪里。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **setLowerMargin(double margin)** 它为轴心的利润率较低（为轴心范围的百分比），并发送一个AxisChangeEvent所有已注册的侦听器。这个方法是从父类ValueAxis继承。 |
| 2 | **setUpperMargin(double margin)** 它设置置于所述轴的上缘（视轴范围的百分比），并发送一个AxisChangeEvent给所有注册的监听器。这种方法也存在于ValueAxis类。 |

## XYLineAndShapeRenderer 类

这是在org.jfree.chart.renderer.xy包下的类，它需要连接数据点与线，并绘制形状，在每个数据点下是可用的。这个渲染器类是专为XYPlot类配合使用。

### 类的构造函数

| S.N. | 构造和描述 |
|---|---|
| 1 | **XYLineAndShapeRenderer()** 它创建了一个新的渲染器有两种线条和形状可见。 |
| 2 | **XYLineAndShapeRenderer** (boolean lines, boolean shapes) 它创建了一个新的渲染与特定的属性。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **setSeriesPaint(int series, java.awt.Paint paint)** 此方法设置用于一系列的涂料，并发送RendererChangeEvent给所有注册的监听器。这个方法是从AbstratRenderer抽象类从渲染器包中JFreeChart的API。 |
| 2 | **setSeriesStroke(int series, java.awt.Stroke stroke)** 此方法设置用于一系列的流程，并发送RendererChangeEvent向所有注册的侦听器。这个方法是从AbstratRenderer抽象类，它是这个包的超类。 |

## XYItemRenderer通用数据集

这是用于使一个单一的（X，Y）格式项在XYPlot接口。org.Jfree.data.general包其具有类和接口，以定义不同类型的数据集来构造图。

## PieDataset

这是作为一个通用的数据集，其中值与键相关联的接口。正如其名称所暗示的，可以使用这个数据集提供数据的饼图。此接口扩展KeyedValues数据集的接口。所有使用此接口的方法取自KeyedValues，Values和数据集的接口。

## DefaultPieDataset 类

这是一个默认的实现类PieDataset接口。

### 类的构造函数

| S.N. | 构造函数和描述 |
| --- | --- |
| 1 | **DefaultPieDataset()** 该构造函数创建一个新的数据集，初始为空。 |
| 2 | **DefaultPieDataset(KeyedValues data)** 它从一个KeyedValues实例复制数据创建了一个新的数据集。 |

### 类方法

| S.N. | 方法及描述 |
| --- | --- |
| 1 | **setValue(java.lang.Comparable key, double value)** 它设置数据值的键，发送DatasetChangeEvent向所有注册的侦听器。 |
| 2 | **setValue(java.lang.Comparable key, java.lang.Number value)** 它设置数据值的键，发送DatasetChangeEvent向所有注册的侦听器。 |

## SeriesException 类

这是一个异常类。它会引发发生在时间序列中数据集的数据的异常。异常是引发上的重复或无效数据的次数。时间序列不能与重复应用，格式必须是有效的。

## DefaultCategoryDataset

这是一个默认的实现类CategoryDataset接口。

### 类的构造函数

| S.N. | 构造函数及描述 |
| --- | --- |
| 1 | **DefaultCategoryDataset()** 此构造函数创建新的空数据集。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **addValue**(double value, java.lang.Comparable rowKey, java.lang.Comparable columnKey) 这种方法增加了一个值，以使用可比的键表。 |
| 2 | **addValue**(java.lang.Number value, java.lang.Comparable rowKey, java.lang.Comparable columnKey) 这种方法增加了一个值的表。 |
| 3 | **setValue**(double value, java.lang.Comparable rowKey, java.lang.Comparable columnKey) 此方法添加或在表中更新的值，并发送aDatasetChangeEvent给所有注册的监听器。 |
| 4 | **setValue**(java.lang.Number value, java.lang.Comparable rowKey, java.lang.Comparable columnKey) 此方法添加或在表中更新的值，并发送DatasetChangeEvent给所有注册的监听器。 |

参见JFreeChart的API，用于各种其他方法和字段的详细信息。

## 序列数据集

系列数据集用于XY图表。该软件包是org.Jfree.data.xy，其中包含类和属于XY图表接口。核心接口是XYDataset。

# XYDataset

这是通过该数据中的（X，Y）的项目的形式可被访问的接口。正如其名称所提示的，可以使用这个数据集服务XY图表。一些在这个接口中的方法都取自SeriesDateset接口。

# XYZDataset

这是通过该数据的形式（x，y，z）的项目可被访问的接口。正如其名称所暗示的，可以使用这个数据集服务XYZ图。一些在这个接口中的方法都取自SeriesDateset。

# XYSeries

这是一类，它代表了在所述形式的零个或多个数据项（x，y）的序列。默认情况下，该系列中的数据项都按升序排列由x值，并重复允许的x值。无论是排序和复制缺省值可以在构造函数中被改变。 Y值可以表示为空值代表缺失值。

### 类构造函数

| S.N. | 构造函数描述 |
|---|---|
| 1 | **XYSeries**(java.lang.Comparable key) 该构造函数创建一个新的空系列。 |
| 2 | **XYSeries**(java.lang.Comparable key, boolean autoSort) 它构造一个新的空系列，具有自动排序标志集的请求，并且重复的值是允许的。 |
| 3 | **XYSeries**(java.lang.Comparable key, boolean autoSort, boolean allowDuplicateXValues) 它构造一个新的xy系列不包含任何数据。 |

### 类方法

| S.N. | 方法描述 |
|---|---|
| 1 | **add(double x, double y)** 这种方法增加了数据项成系列。 |

在上述方法中使用的教程例子。如果想了解其余的方法和字段，请参考JFreeChart的API。

## XYSeriesCollection

XYSeriesCollection类有类似父类AbstractIntervelDataset，AbstractXYDatset，AbstractSeriesDataset和AbstractDataset。一些在这个类中的方法属于这个类的父类。

### 类的构造函数

| S.N. | 构造函数描述 |
|---|---|
| 1 | **XYSeriesCollection()** 它构造一个空的数据集。 |
| 2 | **XYSeriesCollection(XYSeries xyseries)**它构建了一个数据集，并用一个系列的填充。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **addSeries(XYSeries series)** 这种方法增加了一系列的收集和发送DatasetChangeEvent向所有注册的侦听器。 |

参见JFreeChart的API其余的方法和字段。

## Default XYZDataset :

DefaultXYZDataset类都有父类，如AbstractIntervelDataset，AbstractXYDatset，AbstractSeriesDataset，AbstractDataset和AbstractXYZDataset。一些在这个类中的方法属于这一类的父类。

### 类的构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **DefaultXYZDataset()** 它构造一个空的数据集。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **addSeries(java.lang.Comparable seriesKey, double[ ][ ] data )** 该方法增加了一系列的收集和发送DatasetChangeEvent向所有注册的侦听器。 |

请参考JFreeChart的API，其余的方法和字段。

## JFreeCharts的时间序列

该软件包是org.jfree.data.time。该软件包包含用于时间相关的数据的类和接口。

## TimeSeries :

此类表示数据项的期值的形式，其中一段是RegularTimePeriod抽象类，如时间，日，小时，分钟和秒类的一些实例序列。

### 类的构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **TimeSeries(java.lang.Comparable name)** 它创建新的空系列。 |
| 2 | **TimeSeries(java.lang.Comarable name, java.lang.String domain, java.lang.Strin range)** 它会创建一个不包含任何数据的新的时间序列。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **add(RegularTimePeriod period,double value)** 该方法增加了一个新的数据项用以串联。 |

其余的方法和字段参见JFreeChart的API。

# TimeSeriesCollection :

这是作为时间序列的对象的集合的类。这个类实现了XYDataset接口，以及它扩展了IntervelXYDataset接口。这使得它可以方便地收集序列数据对象。

## 类的构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **TimeSeriesCollection()** 它构造一个空的数据集，绑在默认时区。 |
| 2 | **TimeSeriesCollection(TimeSeries** series) 它构造一个包含单个系列（更多可添加），绑在默认时区的数据集。 |
| 3 | **TimeSeriesCollection(TimeSeries** series, java.util.TimeZone zone) 它构造包含单个系列（更可添加），绑定到特定的时间段的数据集。 |
| 4 | **TimeSeriesCollection**(java.util.TimeZone zone) 它构造一个空的数据集时，绑定到特定的时间区。 |

## 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **addSeries(TimeSeries** series) 方法增加了一系列的收集和发送DatasetChangeEvent向所有注册的侦听器。 |

其余的方法和字段请参考JFreeChart的API。

# Second 类:

这个类表示一个特定的日子一秒钟。这个类是不可变的，这是对所有RegularTimePeriod子类的要求。

## 类的构造函数

| S.N. | 构造函数及描述 |
|---|---|
| 1 | **Second()** 它构造一个新的Second，基于系统的日期/时间。 |
| 2 | **Second**(java.util.Date time) 它构造从指定日期/时间和默认时区的新实例。 |
| 3 | **Second**(java.util.Date time, java.util.TimeZone zone, java.util.Locale locale) 它创建基于所提供的时间和时区的新的Second对象。 |
| 4 | **Second**(int second, int minute, int hour, int day, int month, int year)它创建了一个新的Second对象。 |
| 5 | **Second**(int second, Minute minute) 它构建了一个新的Second。 |

### 类方法

| S.N. | 方法及描述 |
|---|---|
| 1 | **getSecond()** 它返回分钟和秒。 |
| 2 | **next()** 它返回当前的下一秒。 |

其余的方法和字段请参考JFreeChart的API。

## JFreeCharts 中的帧:

该软件包是org.jfree.ui。这是包所属JFreeChart的JCommons的API。它包含用于创建预配置的图表框架的实用程序类。

## ApplicationFrame :

这是用于创建简单的应用程序的主框架的基类。帧监听窗口关闭事件，并作出反应，关闭JVM。这是很好的小型演示应用。对于企业应用程序，需要使用一些更稳健的东西。在这个类中的主要核心方法取自Component, Container, Window, Frame 和Jframe类。

### 类构造函数

| S.N. | 构造方法及描述 |
|---|---|
| 1 | **ApplicationFrame(java.lang.String title)** 它会创建一个字符串标题的应用程序框架。 |

这个类有助于创建AWT框架。这就是为什么我们使用这个类作为父类在本教程中的例子的原因。

其采取父类的方法用于打开一个框架，关闭一个框架，改变大小，改变背景或前景颜色和监听器。

## RefineryUtilities :

这是关于用户界面的工具方法的类的集合。

### 类方法

| S.N. | 方法及描述 |
| --- | --- |
| 1 | **centerFrameOnScreen**(java.awt.Window frame) 它定位在屏幕的中间的指定帧。 |

在上述方法中使用的教程例子以外的类，方法和字段参见JFreeChart的API。

# JFreeChart饼图 - JFreeChart教程

在饼图中，每个扇区的弧长成正比它代表的数量。本章演示了如何使用JFreeChart从一个给定的业务数据创建饼图。

## 业务数据

下面的例子描述了移动销售饼图。以下是不同移动品牌和销售(每天单位)列表。

| S.N. | 手机品牌 | 销售(天) |
|---|---|---|
| 1 | Iphone 5S | 20 |
| 2 | Samsung Grand | 20 |
| 3 | MOTO G | 40 |
| 4 | Nokia Lumia | 10 |

## 基于AWT 应用

以下是对从上述给定的信息创建饼图的代码。此代码可以帮助嵌入一个饼图在任何AWT 应用程序。

```
import javax.swing.JPanel;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.data.general.PieDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

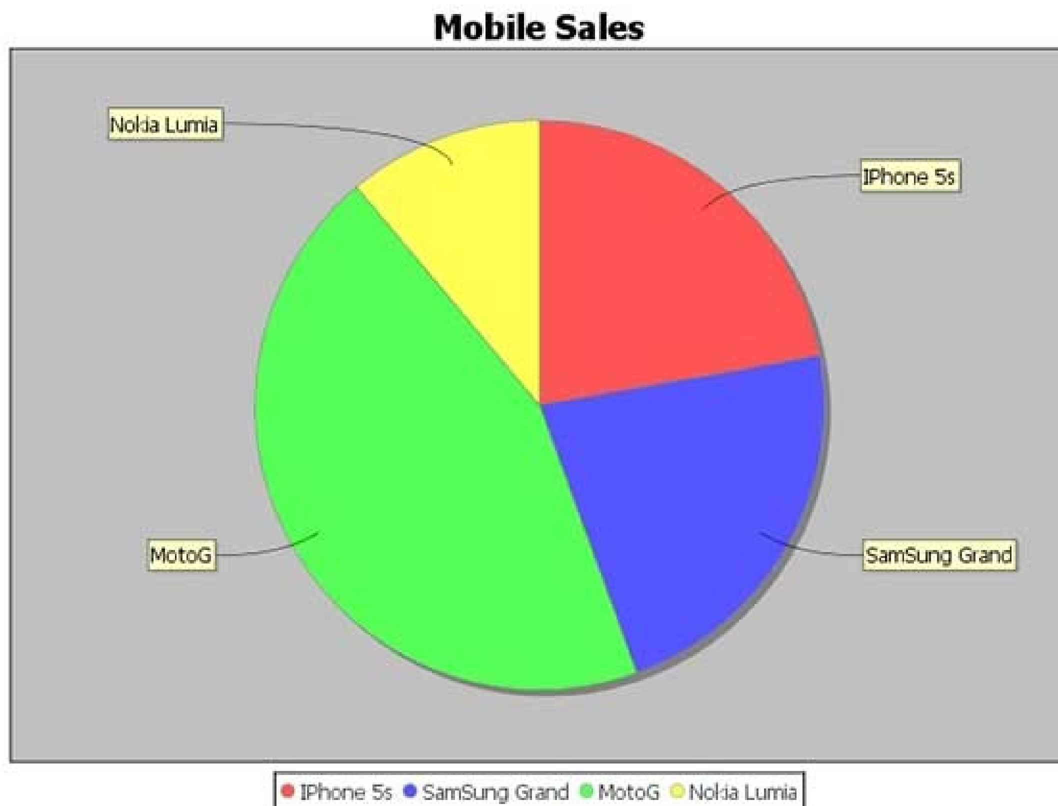
public class PieChart_AWT extends ApplicationFrame
{
   public PieChart_AWT( String title )
   {
      super( title );
      setContentPane(createDemoPanel( ));
   }
   private static PieDataset createDataset( )
   {
      DefaultPieDataset dataset = new DefaultPieDataset( );
      dataset.setValue( "IPhone 5s" , new Double( 20 ) );
      dataset.setValue( "SamSung Grand" , new Double( 20 ) );
      dataset.setValue( "MotoG" , new Double( 40 ) );
      dataset.setValue( "Nokia Lumia" , new Double( 10 ) );
      return dataset;
   }
   private static JFreeChart createChart( PieDataset dataset )
   {
      JFreeChart chart = ChartFactory.createPieChart(
         "Mobile Sales",  // chart title
         dataset,        // data
         true,           // include legend
         true,
         false);

      return chart;
   }
   public static JPanel createDemoPanel( )
   {
      JFreeChart chart = createChart(createDataset( ) );
      return new ChartPanel( chart );
   }
   public static void main( String[ ] args )
   {
      PieChart_AWT demo = new PieChart_AWT( "Mobile Sales" );
      demo.setSize( 560 , 367 );
      RefineryUtilities.centerFrameOnScreen( demo );
      demo.setVisible( true );
   }
}
```

让我们继续上面PieChart_AWT.java文件中的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac PieChart_AWT.java
$java PieChart_AWT
```

如果一切顺利，它会编译并运行生成以下饼图：




如果不需要在应用程序嵌入图表中，那么可以在命令提示符下创建图表的图像。JFreeChart允许以JPG或PNG格式保存图表图像。

## JPEG创建图像

让我们重新写上面的例子，生成命令行JPEG图像。以下是通过JFreeChart库，按要求提供两个API，用它来生成PNG或JPEG图像。

- saveChartAsPNG() - API用来保存图像为PNG格式。
- saveChartAsJPEG() - API用来保存图像JPEG格式。

```
import java.io.*;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

public class PieChart {
   public static void main( String[ ] args ) throws Exception
   {

      DefaultPieDataset dataset = new DefaultPieDataset( );
      dataset.setValue("IPhone 5s", new Double( 20 ) );
      dataset.setValue("SamSung Grand", new Double( 20 ) );
      dataset.setValue("MotoG", new Double( 40 ) );
      dataset.setValue("Nokia Lumia", new Double( 10 ) );

      JFreeChart chart = ChartFactory.createPieChart(
         "Mobile Sales", // chart title
         dataset, // data
         true, // include legend
         true,
         false);

      int width = 640; /* Width of the image */
      int height = 480; /* Height of the image */
      File pieChart = new File( "PieChart.jpeg" );
      ChartUtilities.saveChartAsJPEG( pieChart , chart , width , he
   }
}
```

保持PieChart.java文件中如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac PieChart.java
$java PieChart
```

如果一切顺利，它会编译并运行在当前的目录中创建JPEG图像文件为namedPieChart.jpeg。

# JFreeChart条形图 - JFreeChart教程

本章演示了如何使用JFreeChart从一个给定的业务数据创建条形图。

条形图使用不同的方位(水平或垂直)条，以显示不同类别的比较。图表中的一个轴(域轴)示出了特定的域进行比较，并在另一个轴(范围轴)表示的离散值。

## 业务数据

下面的例子描述了各种汽车用统计柱状图。以下是汽车品牌以及它们的不同特点，我们将展示使用一个条形图的列表：

| 汽车 | 速度 | 用户评价 | 公里数 | 安全性 |
|---|---|---|---|---|
| Fiat | 1.0 | 3.0 | 5.0 | 5.o |
| --- | --- | --- | --- | --- |
| Audi | 5.0 | 6.0 | 10.0 | 4.0 |
| --- | --- | --- | --- | --- |
| Ford | 4.0 | 2.0 | 3.0 | 6.0 |
| --- | --- | --- | --- | --- |

## 基于AWT的应用

以下是对从上述给定的信息创建条形图的代码。此代码可以在AWT的应用程序嵌入一个条形图。

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class BarChart_AWT extends ApplicationFrame
{
   public BarChart_AWT( String applicationTitle , String chartTitle
   {
      super( applicationTitle );
      JFreeChart barChart = ChartFactory.createBarChart(
         chartTitle,
```

```
         "Category",
         "Score",
         createDataset(),
         PlotOrientation.VERTICAL,
         true, true, false);

      ChartPanel chartPanel = new ChartPanel( barChart );
      chartPanel.setPreferredSize(new java.awt.Dimension( 560 , 367
      setContentPane( chartPanel );
   }
   private CategoryDataset createDataset( )
   {
      final String fiat = "FIAT";
      final String audi = "AUDI";
      final String ford = "FORD";
      final String speed = "Speed";
      final String millage = "Millage";
      final String userrating = "User Rating";
      final String safety = "safety";
      final DefaultCategoryDataset dataset =
      new DefaultCategoryDataset( );

      dataset.addValue( 1.0 , fiat , speed );
      dataset.addValue( 3.0 , fiat , userrating );
      dataset.addValue( 5.0 , fiat , millage );
      dataset.addValue( 5.0 , fiat , safety );

      dataset.addValue( 5.0 , audi , speed );
      dataset.addValue( 6.0 , audi , userrating );
      dataset.addValue( 10.0 , audi , millage );
      dataset.addValue( 4.0 , audi , safety );

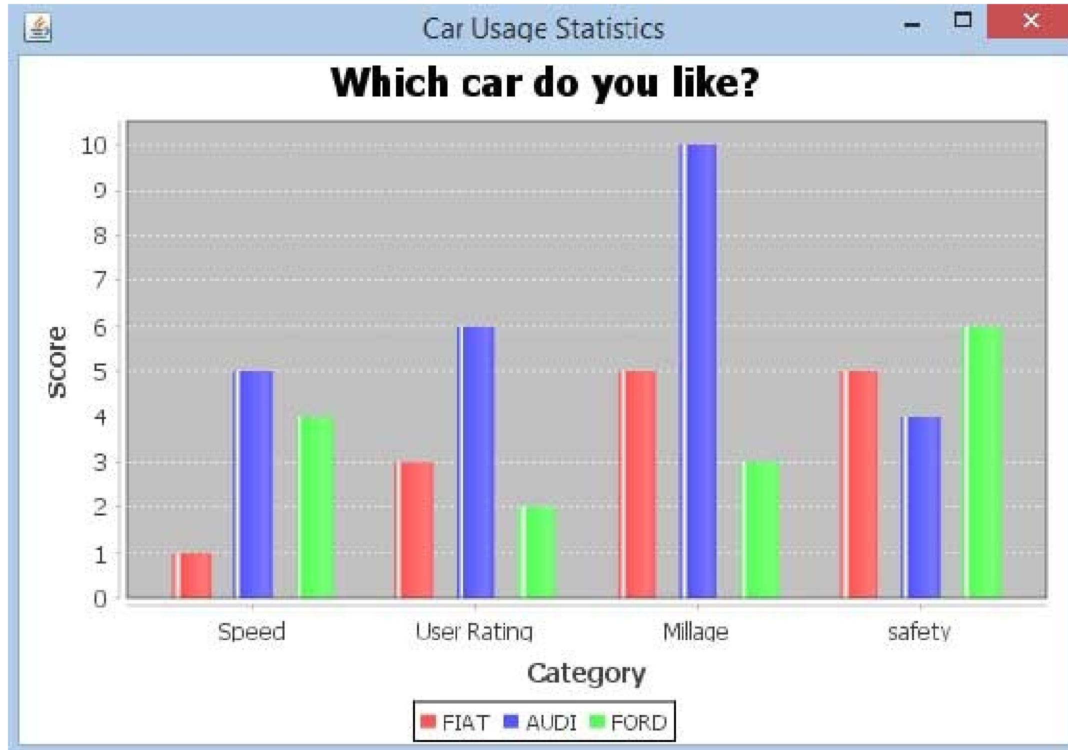
      dataset.addValue( 4.0 , ford , speed );
      dataset.addValue( 2.0 , ford , userrating );
      dataset.addValue( 3.0 , ford , millage );
      dataset.addValue( 6.0 , ford , safety );

      return dataset;
   }
   public static void main( String[ ] args )
   {
      BarChart_AWT chart = new BarChart_AWT("Car Usage Statistics",
      chart.pack( );
      RefineryUtilities.centerFrameOnScreen( chart );
      chart.setVisible( true );
   }
}
```

保持BarChart_AWT.java文件中的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac BarChar_AWT.java
$java BarChart_AWT
```

如果一切顺利，它会编译并运行生成以下条形图：




## JPEG创建图像

让我们重新写上面的例子中，使用命令行生成JPEG图像。

```
import java.io.*;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.chart.ChartUtilities;

public class BarChart
{
   public static void main( String[ ] args )throws Exception
   {

      final String fiat = "FIAT";
      final String audi = "AUDI";
      final String ford = "FORD";
      final String speed = "Speed";
      final String millage = "Millage";
      final String userrating = "User Rating";
      final String safety = "safety";

      final DefaultCategoryDataset dataset = new DefaultCategoryDat

      dataset.addValue( 1.0 , fiat , speed );
      dataset.addValue( 3.0 , fiat , userrating );
      dataset.addValue( 5.0 , fiat , millage );
      dataset.addValue( 5.0 , fiat , safety );

      dataset.addValue( 5.0 , audi , speed );
      dataset.addValue( 6.0 , audi , userrating );
      dataset.addValue( 10.0 , audi , millage );
      dataset.addValue( 4.0 , audi , safety );

      dataset.addValue( 4.0 , ford , speed );
      dataset.addValue( 2.0 , ford , userrating );
      dataset.addValue( 3.0 , ford , millage );
      dataset.addValue( 6.0 , ford , safety );

      JFreeChart barChart = ChartFactory.createBarChart(
         "CAR USAGE STATIStICS",
         "Category", "Score",
         dataset,PlotOrientation.VERTICAL,
         true, true, false);

      int width = 640; /* Width of the image */
      int height = 480; /* Height of the image */
      File BarChart = new File( "BarChart.jpeg" );
      ChartUtilities.saveChartAsJPEG( BarChart , barChart , width ,
   }
}
```

保存BarChart.java文件中如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac BarChart.java
$java BarChart
```

如果一切顺利，它会编译并运行在当前的目录中创建JPEG图像文件namedBarChart.jpeg

# JFreeChart线型图 - JFreeChart教程

线图或折线图来显示信息为一系列由直线段连接的数据点(标记)。线图显示数据在相同的时间频率如何变化。本章从一个给定的业务数据演示如何使用JFreeChart创建线型图。

## 业务数据

下面的示例绘制折线图显示从1970年开始学校在不同年份开通数量。

给定的数据如下：

| 年份 | 学校数量 |
|---|---|
| 1970 | 15 |
| 1980 | 30 |
| 1990 | 60 |
| 2000 | 120 |
| 2013 | 240 |
| 2014 | 300 |

## 基于AWT的应用

以下是对从上述给定的信息创建线型图的代码。此代码可以帮助在AWT的应用程序嵌入一个折线图。

```
import org.jfree.chart.ChartPanel;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

public class LineChart_AWT extends ApplicationFrame
{
   public LineChart_AWT( String applicationTitle , String chartTitl
   {
      super(applicationTitle);
      JFreeChart lineChart = ChartFactory.createLineChart(
         chartTitle,
         "Years","Number of Schools",
         createDataset(),
         PlotOrientation.VERTICAL,
         true,true,false);

      ChartPanel chartPanel = new ChartPanel( lineChart );
      chartPanel.setPreferredSize( new java.awt.Dimension( 560 , 36
      setContentPane( chartPanel );
   }

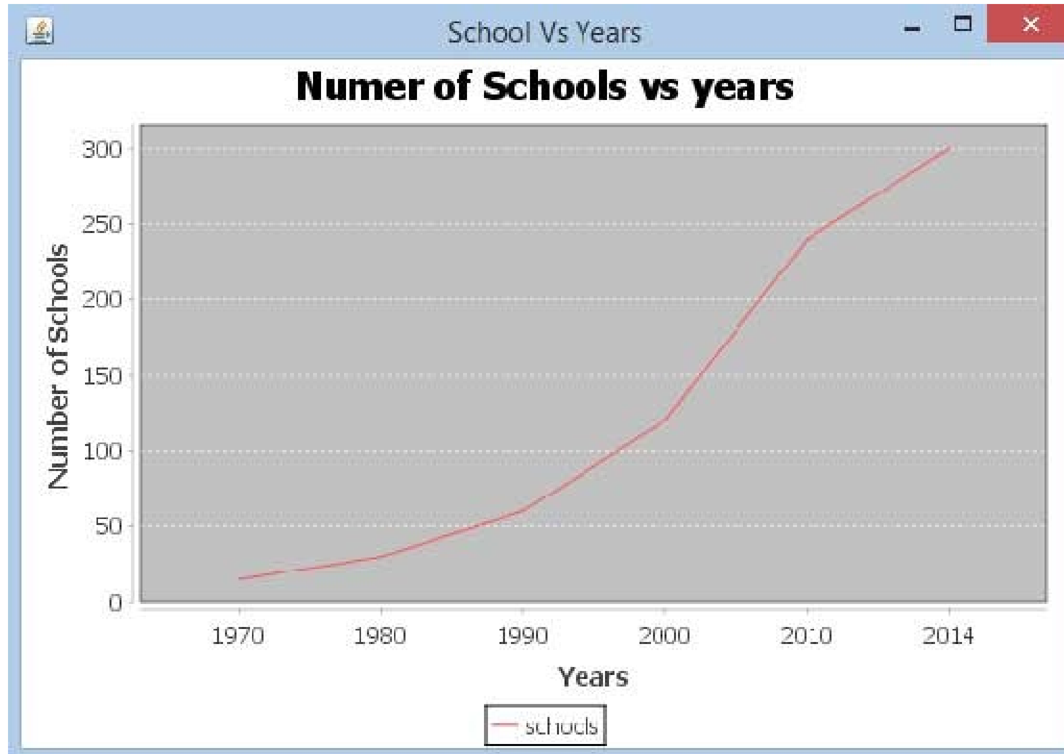
   private DefaultCategoryDataset createDataset( )
   {
      DefaultCategoryDataset dataset = new DefaultCategoryDataset(
      dataset.addValue( 15 , "schools" , "1970" );
      dataset.addValue( 30 , "schools" , "1980" );
      dataset.addValue( 60 , "schools" ,  "1990" );
      dataset.addValue( 120 , "schools" , "2000" );
      dataset.addValue( 240 , "schools" , "2010" );
      dataset.addValue( 300 , "schools" , "2014" );
      return dataset;
   }
   public static void main( String[ ] args )
   {
      LineChart_AWT chart = new LineChart_AWT(
      "School Vs Years" ,
      "Numer of Schools vs years");

      chart.pack( );
      RefineryUtilities.centerFrameOnScreen( chart );
      chart.setVisible( true );
   }
}
```

保存LineChart_AWT.java文件如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac LineChart_AWT.java
$java LineChart_AWT
```

如果一切顺利，它会编译并运行生成以下线图：




## 创建JPEG图像

让我们重新编写上面的例子，在命令行执行生成JPEG图像。

```
import java.io.*;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

public class LineChart
{
   public static void main( String[ ] args ) throws Exception
   {
      DefaultCategoryDataset line_chart_dataset = new DefaultCatego
      line_chart_dataset.addValue( 15 , "schools" , "1970" );
      line_chart_dataset.addValue( 30 , "schools" , "1980" );
      line_chart_dataset.addValue( 60 , "schools" , "1990" );
      line_chart_dataset.addValue( 120 , "schools" , "2000" );
      line_chart_dataset.addValue( 240 , "schools" , "2010" );
      line_chart_dataset.addValue( 300 , "schools" , "2014" );

      JFreeChart lineChartObject = ChartFactory.createLineChart(
         "Schools Vs Years","Year",
         "Schools Count",
         line_chart_dataset,PlotOrientation.VERTICAL,
         true,true,false);

      int width = 640; /* Width of the image */
      int height = 480; /* Height of the image */
      File lineChart = new File( "LineChart.jpeg" );
      ChartUtilities.saveChartAsJPEG(lineChart ,lineChartObject, wi
   }
}
```

让我们保存LineChart.java文件如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac LineChart.java
$java LineChart
```

如果一切顺利，它将编译和执行在当前的目录中创建JPEG图像文件namedLineChart.jpeg。

# JFreeChart XY图 - JFreeChart教程

在xy图(散点图)是根据一个数据系列组成的x和y值的列表。每个值对(x，y)是坐标系中的一个点。这里1值确定水平(X)位置，而另一个确定垂直(Y)位置。本章演示了如何使用JFreeChart从一个给定的业务数据创建XY图表。

## 业务数据

考虑这种情况，我们要创建一个XY图表所有主要浏览器的一个例子。在这里，不同的性能分数是从不同类型的人们聚集，如下所示：

| Firefox | Category(X) | Score(Y) |
|---|---|---|
| 1.0 | 1.0 | |
| 2.0 | 4.0 | |
| 3.0 | 3.0 | |

| Chrome | Category(X) | Score(Y) |
|---|---|---|
| 1.0 | 4.0 | |
| 2.0 | 5.0 | |
| 3.0 | 6.0 | |

| IE | Category(X) | Score(Y) |
|---|---|---|
| 3.0 | 4.0 | |
| 4.0 | 5.0 | |
| 5.0 | 4.0 | |

## 基于AWT的应用

以下是对从上述给定的信息创建XY图表的代码。此代码可以在AWT应用程序嵌入一个XY图表。

```
import java.awt.Color;
import java.awt.BasicStroke;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
```

```
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;

public class XYLineChart_AWT extends ApplicationFrame
{
   public XYLineChart_AWT( String applicationTitle, String chartTit
   {
      super(applicationTitle);
      JFreeChart xylineChart = ChartFactory.createXYLineChart(
         chartTitle ,
         "Category" ,
         "Score" ,
         createDataset() ,
         PlotOrientation.VERTICAL ,
         true , true , false);

      ChartPanel chartPanel = new ChartPanel( xylineChart );
      chartPanel.setPreferredSize( new java.awt.Dimension( 560 , 36
      final XYPlot plot = xylineChart.getXYPlot( );
      XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer(
      renderer.setSeriesPaint( 0 , Color.RED );
      renderer.setSeriesPaint( 1 , Color.GREEN );
      renderer.setSeriesPaint( 2 , Color.YELLOW );
      renderer.setSeriesStroke( 0 , new BasicStroke( 4.0f ) );
      renderer.setSeriesStroke( 1 , new BasicStroke( 3.0f ) );
      renderer.setSeriesStroke( 2 , new BasicStroke( 2.0f ) );
      plot.setRenderer( renderer );
      setContentPane( chartPanel );
   }

   private XYDataset createDataset( )
   {
      final XYSeries firefox = new XYSeries( "Firefox" );
      firefox.add( 1.0 , 1.0 );
      firefox.add( 2.0 , 4.0 );
      firefox.add( 3.0 , 3.0 );
      final XYSeries chrome = new XYSeries( "Chrome" );
      chrome.add( 1.0 , 4.0 );
      chrome.add( 2.0 , 5.0 );
      chrome.add( 3.0 , 6.0 );
      final XYSeries iexplorer = new XYSeries( "InternetExplorer" )
      iexplorer.add( 3.0 , 4.0 );
      iexplorer.add( 4.0 , 5.0 );
      iexplorer.add( 5.0 , 4.0 );
      final XYSeriesCollection dataset = new XYSeriesCollection( );
      dataset.addSeries( firefox );
      dataset.addSeries( chrome );
      dataset.addSeries( iexplorer );
```

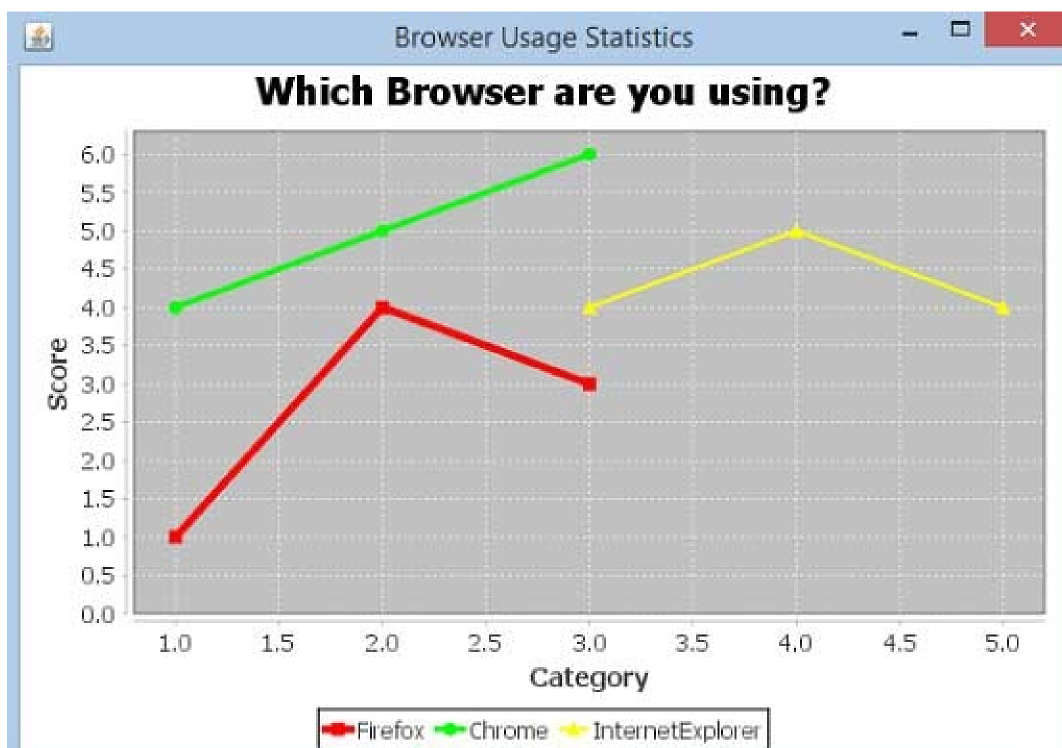
```
        return dataset;
    }

    public static void main( String[ ] args )
    {
        XYLineChart_AWT chart = new XYLineChart_AWT("Browser Usage St
        chart.pack( );
        RefineryUtilities.centerFrameOnScreen( chart );
        chart.setVisible( true );
    }
}
```

让我们保存XYLineChart_AWT.java文件如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac XYLineChart_AWT.java
$java XYLineChart_AWT
```

如果一切顺利，它会编译并运行生成以下XY图：




# 创建JPEG图像

让我们重新编写上面的例子，在命令行生成JPEG图像。

```
import java.io.*;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.xy.XYSeries;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.chart.ChartUtilities;

public class XYLineChart_image
{
   public static void main( String[ ] args )throws Exception
   {
      final XYSeries firefox = new XYSeries( "Firefox" );
      firefox.add( 1.0 , 1.0 );
      firefox.add( 2.0 , 4.0 );
      firefox.add( 3.0 , 3.0 );
      final XYSeries chrome = new XYSeries( "Chrome" );
      chrome.add( 1.0 , 4.0 );
      chrome.add( 2.0 , 5.0 );
      chrome.add( 3.0 , 6.0 );
      final XYSeries iexplorer = new XYSeries( "InternetExplorer" )
      iexplorer.add( 3.0 , 4.0 );
      iexplorer.add( 4.0 , 5.0 );
      iexplorer.add( 5.0 , 4.0 );
      final XYSeriesCollection dataset = new XYSeriesCollection( );
      dataset.addSeries( firefox );
      dataset.addSeries( chrome );
      dataset.addSeries( iexplorer );

      JFreeChart xylineChart = ChartFactory.createXYLineChart(
         "Browser usage statastics",
         "Category",
         "Score",
         dataset,
         PlotOrientation.VERTICAL,
         true, true, false);

      int width = 640; /* Width of the image */
      int height = 480; /* Height of the image */
      File XYChart = new File( "XYLineChart.jpeg" );
      ChartUtilities.saveChartAsJPEG( XYChart, xylineChart, width,
   }
}
```

让我们保存在XYLineChart_image.java文件如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac XYLineChart_image.java
$java XYLineChart_image
```

如果一切顺利，它会编译并运行在当前的目录中创建JPEG图像文件
namedXYLineChart.jpeg。

# JFreeChart 3D饼图/条形图 - JFreeChart教程

三维/3D图表是那些显示在一个三维格式。可以使用这些图表来提供更好的显示效果和清晰的信息。三维/3D饼图是饼图另外一个不错的3D效果。 3D效果可以通过添加一些额外的代码来实现，它会创建一个饼图3D效果。

## 3D饼图

请看下面的例子来描述移动销售三维饼图。以下是不同移动品牌和销售(每天)名单。

| S.N. | 手机品牌 | 销量 (天) |
|---|---|---|
| 1 | Iphone 5S | 20 |
| 2 | Samsung Grand | 20 |
| 3 | MOTO G | 40 |
| 4 | Nokia Lumia | 10 |

以下是对从上述给定的信息创建3D饼图的代码。此代码可以帮助嵌入一个饼图在AWT应用程序中。

```
import java.io.*;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PiePlot3D;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.chart.ChartUtilities;

public class PieChart3D
{
   public static void main( String[ ] args )throws Exception
   {
      DefaultPieDataset dataset = new DefaultPieDataset( );
      dataset.setValue( "IPhone 5s" , new Double( 20 ) );
      dataset.setValue( "SamSung Grand" , new Double( 20 ) );
      dataset.setValue( "MotoG" , new Double( 40 ) );
      dataset.setValue( "Nokia Lumia" , new Double( 10 ) );

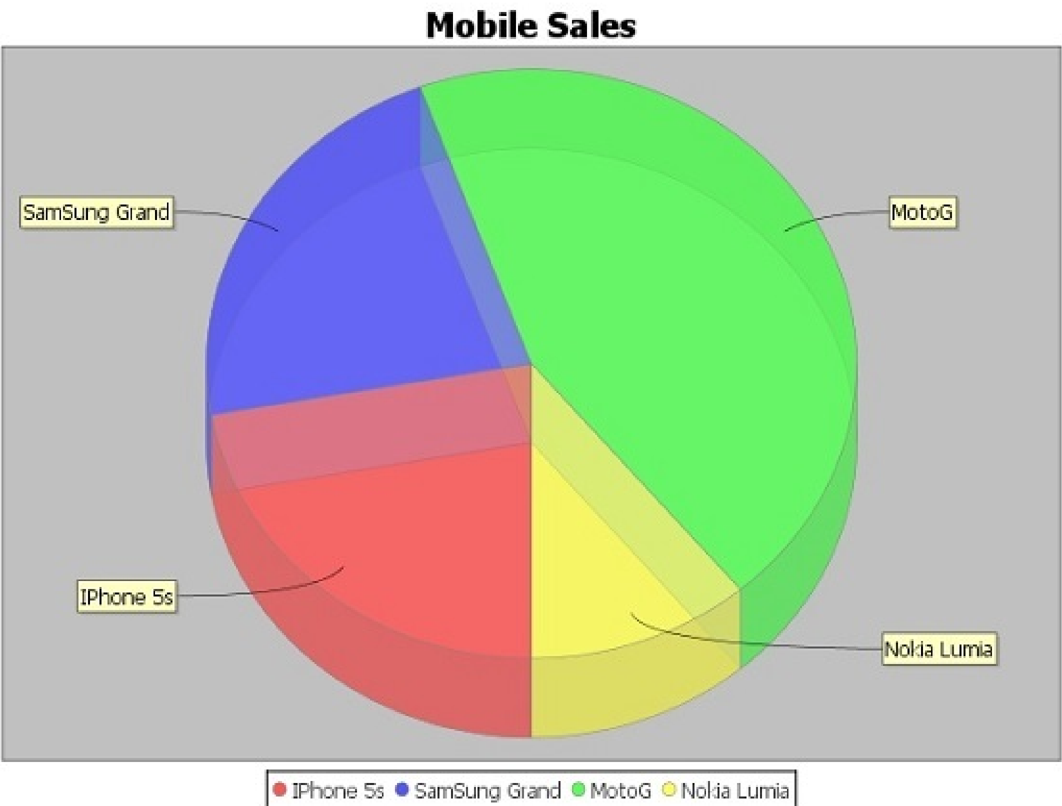
      JFreeChart chart = ChartFactory.createPieChart3D(
         "Mobile Sales" ,  // chart title
         dataset ,         // data
         true ,            // include legend
         true,
         false);

      final PiePlot3D plot = ( PiePlot3D ) chart.getPlot( );
      plot.setStartAngle( 270 );
      plot.setForegroundAlpha( 0.60f );
      plot.setInteriorGap( 0.02 );
      int width = 640; /* Width of the image */
      int height = 480; /* Height of the image */
      File pieChart3D = new File( "pie_Chart3D.jpeg" );
      ChartUtilities.saveChartAsJPEG( pieChart3D , chart , width ,
   }
}
```

让我们保存在PieChart3D.java文件如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac PieChart3D.java
$java PieChart3D
```

如果一切顺利，它会编译并运行，以创建一个名为PieChart3D.jpeg 如以下3D饼图JPEG图像文件：




## 3D条形图

三维条形图是一样的条形图另外一个不错的3D效果。 3D效果可以通过添加一些额外的代码来实现，它会创建一个柱形图3D效果。看看下面的例子，描绘各种汽车的统计数据3D条形图。以下是汽车品牌以及它们的不同特点，我们将展示使用一个条形图的列表：

| Car | Speed | User Rating | Millage | Safety |
|---|---|---|---|---|
| FIAT | 1.0 | 3.0 | 5.0 | 5.o |
| --- | --- | --- | --- | --- |
| AUDI | 5.0 | 6.0 | 10.0 | 4.0 |
| --- | --- | --- | --- | --- |
| FORD | 4.0 | 2.0 | 3.0 | 6.0 |
| --- | --- | --- | --- | --- |

下面的代码从上面给出的信息来创建3D条形图。此代码可以帮助嵌入一个条形图在AWT应用程序中。

```
import java.io.*;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.chart.ChartUtilities;

public class BarChart3D
{
   public static void main( String[ ] args )throws Exception
   {
      final String fait = "FAIT";
      final String audi = "AUDI";
      final String ford = "FORD";
      final String speed = "Speed";
      final String popular = "Popular";
      final String mailage = "Mailage";
      final String userrating = "User Rating";
      final String safty = "safty";
      final DefaultCategoryDataset dataset = new DefaultCategoryDat

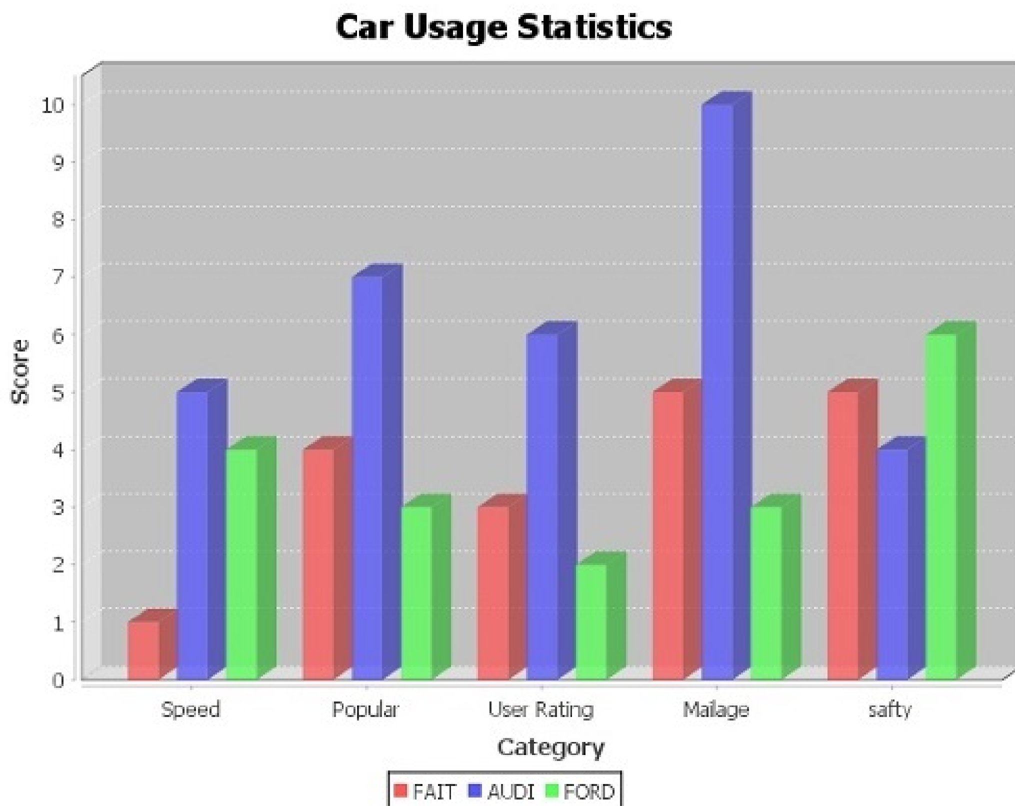
      dataset.addValue( 1.0 , fait , speed );
      dataset.addValue( 4.0 , fait , popular );
      dataset.addValue( 3.0 , fait , userrating );
      dataset.addValue( 5.0 , fait , mailage );
      dataset.addValue( 5.0 , fait , safty );
      dataset.addValue( 5.0 , audi , speed );
      dataset.addValue( 7.0 , audi , popular );
      dataset.addValue( 6.0 , audi , userrating );
      dataset.addValue( 10.0 , audi , mailage );
      dataset.addValue( 4.0 , audi , safty );
      dataset.addValue( 4.0 , ford , speed );
      dataset.addValue( 3.0 , ford , popular );
      dataset.addValue( 2.0 , ford , userrating );
      dataset.addValue( 3.0 , ford , mailage );
      dataset.addValue( 6.0 , ford , safty );
      JFreeChart barChart = ChartFactory.createBarChart3D(
         "Car Usage Statistics",
         "Category",
         "Score",
         dataset,
         PlotOrientation.VERTICAL,
         true, true, false);

      int width = 640; /* Width of the image */
      int height = 480; /* Height of the image */
      File barChart3D = new File( "barChart3D.jpeg" );
      ChartUtilities.saveChartAsJPEG( barChart3D, barChart, width,
   }
}
```

让我们保存在BarChart3D.java文件如上面的Java代码，然后从命令提示符下编译并运行它，如下所示：

```
$javac BarChart3D.java
$java BarChart3
```

如果一切正常，它会编译并运行创建的JPEG图像fileBarChart3D.jpeg，具有下列3D条形图：



# JFreeChart气泡图表 - JFreeChart教程

本章演示如何使用JFreeChart从一个给定的业务数据创建气泡图表。使用气泡图显示在三维方式的信息。气泡绘制在其中(x，y)坐标相交的地方。气泡的大小被认为是范围或X和Y轴的数量。

## 业务数据

考虑不同的人的年龄，体重和工作能力可能不太相同。能力可以视为对该被绘制为图表中的气泡的小时数。

| AGE | 30 | 40 | 50 | 60 | 70 | 80 |
|---|---|---|---|---|---|---|
| 10 | 4 | WORK | | | | |
| 20 | 5 | | | | | |
| 30 | 10 | | | | | |
| 40 | 8 | | | | | |
| 50 | 9 | | | | | |
| 60 | 6 | | | | | |

## 基于AWT的应用

以下是对从上述给定的信息创建气泡图表的代码。此代码可以帮助嵌入一个气泡图在AWT应用程序。

```
import java.awt.Color;
import java.awt.Dimension;
import javax.swing.JPanel;
import org.jfree.chart.*;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.data.xy.DefaultXYZDataset;
import org.jfree.data.xy.XYZDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class BubbleChart_AWT extends ApplicationFrame
{
   public BubbleChart_AWT( String s )
```

```
    {
       super( s );
       JPanel jpanel = createDemoPanel( );
       jpanel.setPreferredSize(new Dimension( 560 , 370 ) );
       setContentPane( jpanel );
    }

    private static JFreeChart createChart( XYZDataset xyzdataset )
    {
       JFreeChart jfreechart = ChartFactory.createBubbleChart(
          "AGE vs WEIGHT vs WORK",
          "Weight",
          "AGE",
          xyzdataset,
          PlotOrientation.HORIZONTAL,
          true, true, false);

       XYPlot xyplot = ( XYPlot )jfreechart.getPlot( );
       xyplot.setForegroundAlpha( 0.65F );
       XYItemRenderer xyitemrenderer = xyplot.getRenderer( );
       xyitemrenderer.setSeriesPaint( 0 , Color.blue );
       NumberAxis numberaxis = ( NumberAxis )xyplot.getDomainAxis( )
       numberaxis.setLowerMargin( 0.2 );
       numberaxis.setUpperMargin( 0.5 );
       NumberAxis numberaxis1 = ( NumberAxis )xyplot.getRangeAxis( )
       numberaxis1.setLowerMargin( 0.8 );
       numberaxis1.setUpperMargin( 0.9 );

       return jfreechart;
    }

    public static XYZDataset createDataset( )
    {
       DefaultXYZDataset defaultxyzdataset = new DefaultXYZDataset()

       double ad[ ] = { 30 , 40 , 50 , 60 , 70 , 80 };
       double ad1[ ] = { 10 , 20 , 30 , 40 , 50 , 60 };
       double ad2[ ] = { 4 , 5 , 10 , 8 , 9 , 6 };
       double ad3[][] = { ad , ad1 , ad2 };
       defaultxyzdataset.addSeries( "Series 1" , ad3 );

       return defaultxyzdataset;
    }

    public static JPanel createDemoPanel( )
    {
       JFreeChart jfreechart = createChart( createDataset( ) );
       ChartPanel chartpanel = new ChartPanel( jfreechart );

       chartpanel.setDomainZoomable( true );
       chartpanel.setRangeZoomable( true );

       return chartpanel;
```

```
    }

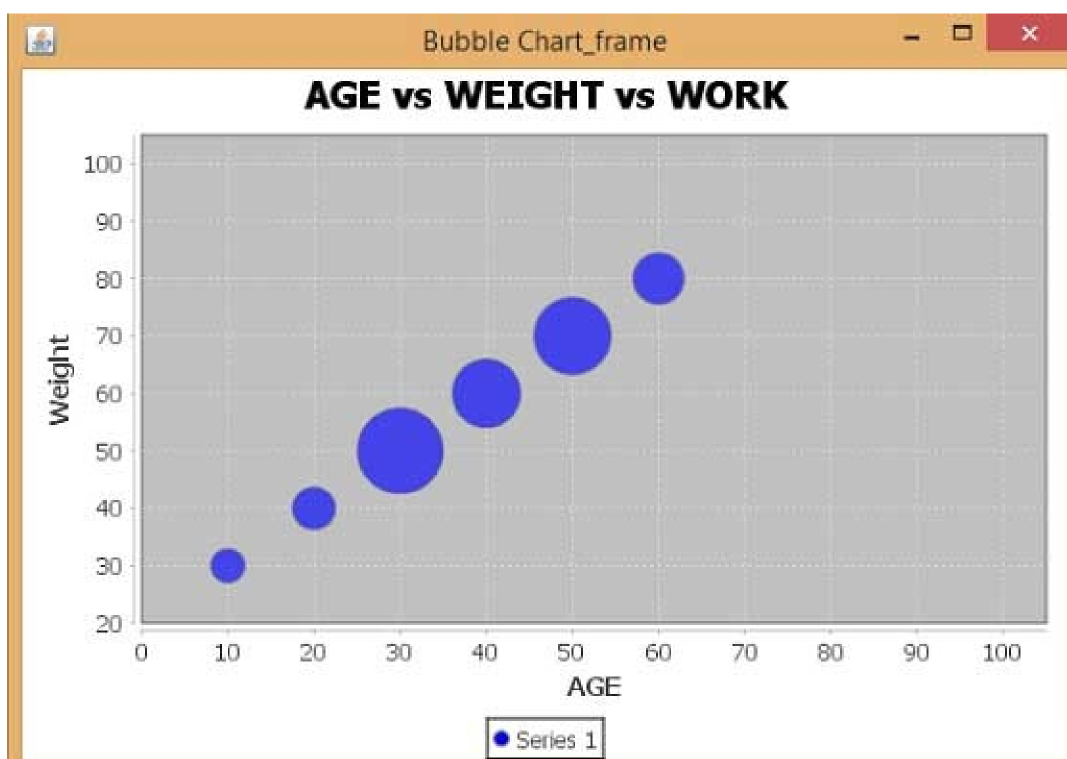
    public static void main( String args[ ] )
    {
       BubbleChart_AWT bubblechart = new BubbleChart_AWT( "Bubble Ch

       bubblechart.pack( );
       RefineryUtilities.centerFrameOnScreen( bubblechart );
       bubblechart.setVisible( true );
    }
 }
```

让我们保存上面的Java代码在BubbleChart_AWT.java文件，然后从命令提示符下编译并运行它，如下所示：

```
$javac BubbleChart_AWT.java
$java BubbleChart_AW
```

如果一切顺利，它会编译并运行生成以下气泡图：




## 创建JPEG图像

让我们重新编写上面的例子，在命令行生成JPEG图像。

```
import java.io.*;
import java.awt.Color;
import org.jfree.chart.*;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.data.xy.DefaultXYZDataset;
import org.jfree.chart.ChartUtilities;

public class BubbleChart_image
{
   public static void main( String args[ ] )throws Exception
   {
      DefaultXYZDataset defaultxyzdataset = new DefaultXYZDataset(
      double ad[ ] = { 30 , 40 , 50 , 60 , 70 , 80 };
      double ad1[ ] = { 10 , 20 , 30 , 40 , 50 , 60 };
      double ad2[ ] = { 4 , 5 , 10 , 8 , 9 , 6 };
      double ad3[ ][ ] = { ad , ad1 , ad2 };
      defaultxyzdataset.addSeries( "Series 1" , ad3 );

      JFreeChart jfreechart = ChartFactory.createBubbleChart(
         "AGE vs WEIGHT vs WORK",
         "Weight",
         "AGE",
         defaultxyzdataset,
         PlotOrientation.HORIZONTAL,
         true, true, false);

      XYPlot xyplot = ( XYPlot )jfreechart.getPlot( );
      xyplot.setForegroundAlpha( 0.65F );
      XYItemRenderer xyitemrenderer = xyplot.getRenderer( );
      xyitemrenderer.setSeriesPaint( 0 , Color.blue );
      NumberAxis numberaxis = ( NumberAxis )xyplot.getDomainAxis( 
      numberaxis.setLowerMargin( 0.2 );
      numberaxis.setUpperMargin( 0.5 );
      NumberAxis numberaxis1 = ( NumberAxis )xyplot.getRangeAxis( 
      numberaxis1.setLowerMargin( 0.8 );
      numberaxis1.setUpperMargin( 0.9 );

      int width = 560; /* Width of the image */
      int height = 370; /* Height of the image */
      File bubbleChart = new File("BubbleChart.jpeg");
      ChartUtilities.saveChartAsJPEG(bubbleChart,jfreechart,width,h
   }
}
```

让我们保存上面的Java代码在BubbleChart_image.java文件，然后从命令提示符下编译并运行它，如下所示：

```
$javac BubbleChart_image.java
$java BubbleChart_image
```

如果一切顺利，它会编译并运行在当前的目录中创建的JPEG图像文件namedBubbleChart.jpeg。

# JFreeChart时序图 - JFreeChart教程

时序图表显示的数据点在相等的时间间隔序列变化。本章演示了如何从一个给定的业务数据使用JFreeChart，建立时序图。

## 业务数据

让我们考虑通过使用标准 Java API 的 Math.random()产生的各种随机数。我们使用这些数字产生一个时间序列图。可以生成用于发生错误的总数在自己的网站给定的时间间隔类似的图表。

## 基于AWT的应用

下面是创建由Math.random()产生的数字时间的代码在给定时间内的序列图。

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.SeriesException;
import org.jfree.data.time.Second;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.data.xy.XYDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class TimeSeries_AWT extends ApplicationFrame
{
   public TimeSeries_AWT( final String title )
   {
      super( title );
      final XYDataset dataset = createDataset( );
      final JFreeChart chart = createChart( dataset );
      final ChartPanel chartPanel = new ChartPanel( chart );
      chartPanel.setPreferredSize( new java.awt.Dimension( 560 , 37
      chartPanel.setMouseZoomable( true , false );
      setContentPane( chartPanel );
   }

   private XYDataset createDataset( )
   {
      final TimeSeries series = new TimeSeries( "Random Data" );
      Second current = new Second( );
      double value = 100.0;
      for (int i = 0; i < 4000; i++)
      {
```

```
         try
         {
            value = value + Math.random( ) - 0.5;
            series.add(current, new Double( value ) );
            current = ( Second ) current.next( );
         }
         catch ( SeriesException e )
         {
            System.err.println("Error adding to series");
         }
      }

      return new TimeSeriesCollection(series);
   }

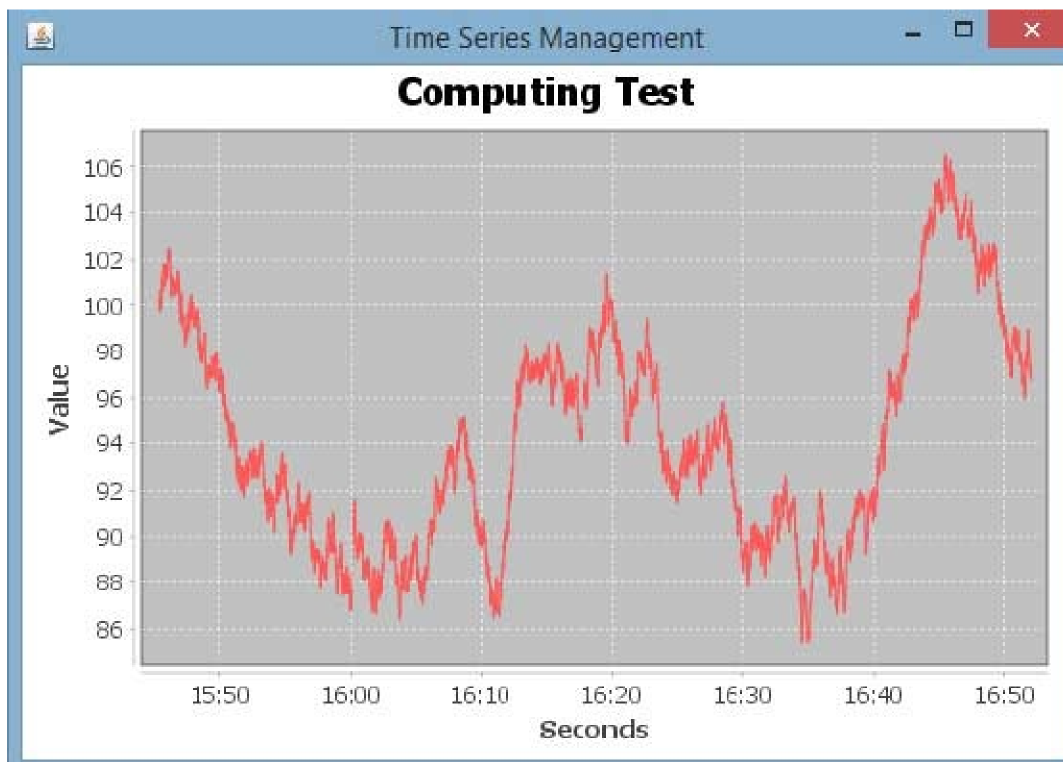
   private JFreeChart createChart( final XYDataset dataset )
   {
      return ChartFactory.createTimeSeriesChart(
      "Computing Test",
      "Seconds",
      "Value",
      dataset,
      false,
      false,
      false);
   }

   public static void main( final String[ ] args )
   {
      final String title = "Time Series Management";
      final TimeSeries_AWT demo = new TimeSeries_AWT( title );
      demo.pack( );
      RefineryUtilities.positionFrameRandomly( demo );
      demo.setVisible( true );
   }
}
```

保存上面的Java代码在TimeSeries_AWT.java文件，然后从命令提示符下编译并运行它，如下所示：

```
$javac TimeSeries_AWT.java
$java TImeSeries_AWT
```

如果一切顺利，它会编译并运行生成以下时序图：




# 创建JPEG图像

让我们重新编写上面的例子，在命令行生成JPEG图像。

```
import java.io.*;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.SeriesException;
import org.jfree.data.time.Second;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.data.xy.XYDataset;
import org.jfree.chart.ChartUtilities;

public class TimeSeriesChart
{
   public static void main( final String[ ] args )throws Exception
   {
      final TimeSeries series = new TimeSeries( "Random Data" );
      Second current = new Second();
      double value = 100.0;
      for ( int i = 0 ; i < 4000 ; i++ )
      {
         try
         {
            value = value + Math.random( ) - 0.5;
            series.add( current , new Double( value ) );
            current = ( Second ) current.next( );
         }
         catch ( SeriesException e )
         {
            System.err.println( "Error adding to series" );
         }
      }
      final XYDataset dataset=( XYDataset )new TimeSeriesCollectio
      JFreeChart timechart = ChartFactory.createTimeSeriesChart(
         "Computing Test",
         "Seconds",
         "Value",
         dataset,
         false,
         false,
         false);

      int width = 560; /* Width of the image */
      int height = 370; /* Height of the image */
      File timeChart = new File( "TimeChart.jpeg" );
      ChartUtilities.saveChartAsJPEG( timeChart, timechart, width,
   }
}
```

继续上面的Java代码保存在TimeSeriesChart.java文件中，然后从命令提示符下编译并运行它，如下所示：

```
$javac TimeSeriesChart.java
$java TimeSeriesChart
```

如果一切正常，它会编译并运行在当前的目录中创建JPEG图像文件TimeChart.jpeg文件。

# JFreeChart文件接口 - JFreeChart教程

到目前为止，我们学习了如何使用静态数据的创建不同类型的JFreeChart API图表。但在生产环境中，数据被设置在文本文件的形式与一个预定义的格式，或者其直接来自数据库。

本章将解释如何我们可以看到从一个给定的位置给定的文本文件，一个简单的数据，然后使用JFreeChart创建图表。

## 业务数据

假设我们有一个文件名为mobile.txt，含有一个简单的逗号分隔不同的移动品牌和销售(每天单位)(，)：

```
Iphone 5S, 20
Samsung Grand, 20
MOTO G, 40  Nokia
Lumia, 10
```

## 基于文件图表生成

下面是基于文件 mobile.txt 提供的信息来创建一个饼图的代码：

```
import java.io.*;
import java.util.StringTokenizer;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

public class PieChart_File
{
   public static void main( String[ ] args )throws Exception
   {
      String mobilebrands[ ] = {
        "IPhone 5s" ,
        "SamSung Grand" ,
        "MotoG" ,
        "Nokia Lumia"
      };

      InputStream in = new FileInputStream( new File( "C:/temp/test
      BufferedReader reader = new BufferedReader(new InputStreamRea
      StringBuilder out = new StringBuilder();
      String line;
      DefaultPieDataset dataset = new DefaultPieDataset();

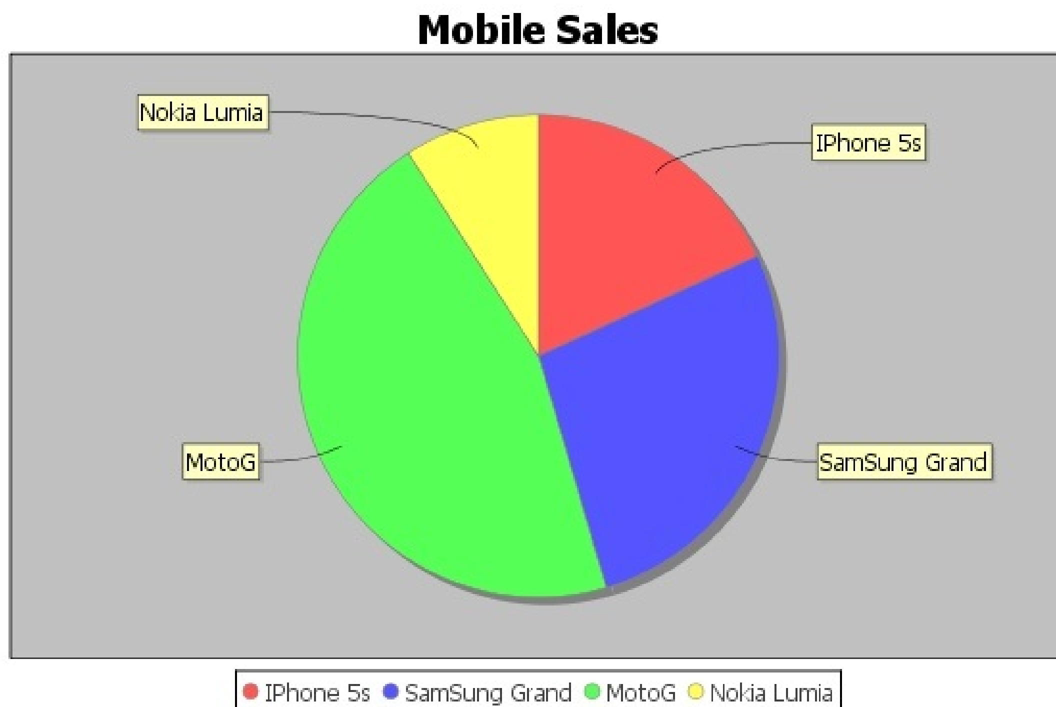
      while (( line = reader.readLine() ) != null )
      {
         out.append( line );
      }
      StringTokenizer s = new StringTokenizer( out.toString(), ","
      int i=0;
      while( s.hasMoreTokens( ) && ( mobilebrands [i] != null ) )
      {
         dataset.setValue(mobilebrands[i], Double.parseDouble( s.ne
         i++;
      }
      JFreeChart chart = ChartFactory.createPieChart(
         "Mobile Sales",  // chart title
         dataset,         // data
         true,            // include legend
         true,
         false);

      int width = 560; /* Width of the image */
      int height = 370; /* Height of the image */
      File pieChart = new File( "pie_Chart.jpeg" );
      ChartUtilities.saveChartAsJPEG( pieChart, chart, width, heigh
   }
}
```

让我们保存上面的Java代码在PieChart_File.java文件，然后从命令提示符下编译并运行它，如下所示：

```
$javac PieChart_File.java
$java PieChart_File
```

如果一切顺利，它会编译并运行以创建包含下面的图表名为PieChart.jpegthat JPEG图像文件。



# JFreeChart数据库接口 - JFreeChart教程

本章介绍如何从数据库表中读取简单的数据，然后 JFreeChart 使用这些数据来创建图表。

## 业务数据

考虑到我们有如下的 MySQL 表 mobile_tbl(mobile_brand VARCHAR(100)NOT NULL，unit_sale INT NO NULL);

考虑这个表含有以下记录：

| 手机品牌 | 销售单位 |
|---|---|
| IPhone5S | 20 |
| Samsung Grand | 20 |
| MotoG | 40 |
| Nokia Lumia | 10 |

## 使用数据库数据生成图表

下面是创建一个基于MySQL数据库 mobile_tbl表 在数据库实例 TEST_DB 中提供的信息饼图的代码。根据需要，可以使用任何其他数据库。

```
import java.io.*;
import java.sql.*;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

public class PieChart_DB
{
   public static void main( String[ ] args )throws Exception
   {
      String mobilebrands[] = {
      "IPhone 5s",
      "SamSung Grand",
      "MotoG",
      "Nokia Lumia"
      };

      /* Create MySQL Database Connection */
      Class.forName( "com.mysql.jdbc.Driver" );
      Connection connect = DriverManager.getConnection(
      "jdbc:mysql://localhost:3306/jf_testdb" ,
      "root",
      "root123");

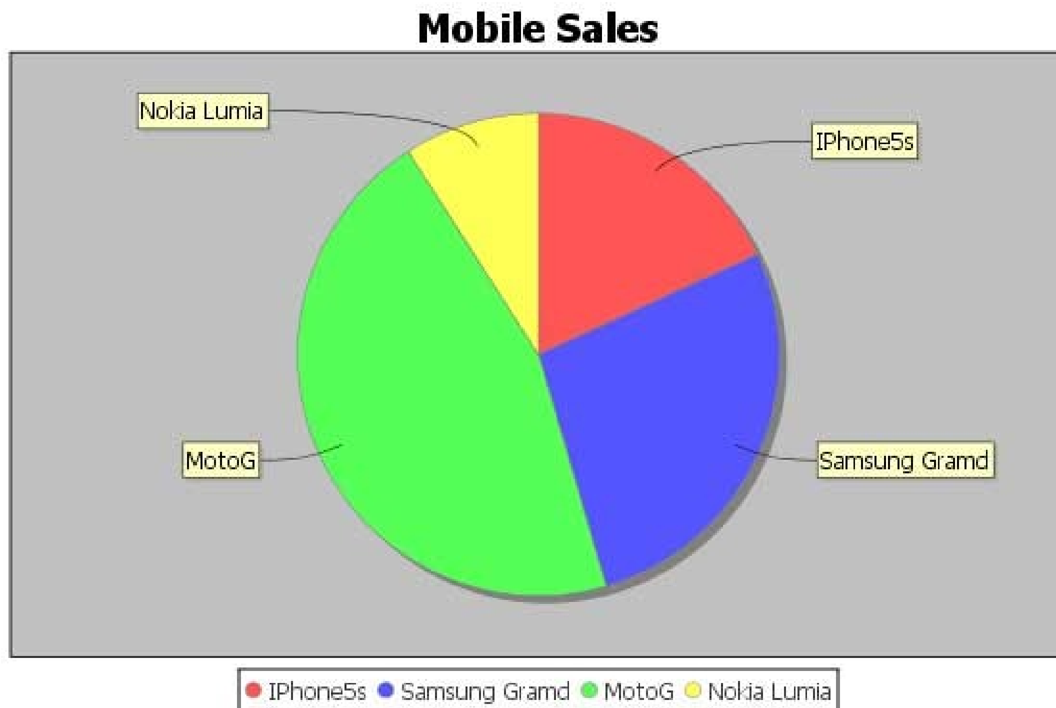
      Statement statement = connect.createStatement( );
      ResultSet resultSet = statement.executeQuery("select * from c
      DefaultPieDataset dataset = new DefaultPieDataset( );
      while( resultSet.next( ) )
      {
         dataset.setValue(
         resultSet.getString( "brandname" ) ,
         Double.parseDouble( resultSet.getString( "datavalue" )));
      }
      JFreeChart chart = ChartFactory.createPieChart(
         "Mobile Sales",  // chart title
         dataset,         // data
         true,            // include legend
         true,
         false );

      int width = 560; /* Width of the image */
      int height = 370; /* Height of the image */
      File pieChart = new File( "Pie_Chart.jpeg" );
      ChartUtilities.saveChartAsJPEG( pieChart , chart , width , he
   }
}
```

让我们保存上面的Java代码在PieChart_DB.java文件，然后从命令提示符下编译并运行它，如下所示：

```
$javac PieChart_DB.java
$java PieChart_DB
```

如果一切顺利，它会编译并运行并创建一个名为Pie_Chart.jpeghavingJPEG图像文件，如下图。



# JMeter教程

## JMeter是什么？

JMeter是一个软件，使负载测试或业绩为导向的业务（功能）测试不同的协议或技术。 Apache软件基金会的Stefano Mazzocchi JMeter的最初的开发。他写道：它主要对 Apache JServ（现在称为如Apache Tomcat项目）的性能进行测试。Apache后来重新设计JMeter 增强的图形用户界面和添加功能测试能力。

这是一个具有图形界面，使用Swing 图形API 的 Java 桌面应用程序，因此可以运行在任何环境/工作站接受一个Java 虚拟机，例如：在Windows，Linux，MAC等。

JMeter 支持的协议是：

- web：HTTP，HTTPS站点的Web1.0的Web 2.0 (ajax, flex and flex-ws-amf)
- Web Services: SOAP / XML-RPC
- 通过JDBC驱动程序的数据库
- 目录: LDAP
- 面向消息的服务通过JMS
- Service: POP3, IMAP, SMTP
- FTP 服务

## JMeter 特点

以下是一些 JMeter 的特点：

- 它是免费的。开放源码软件。
- 它具有简单，直观的图形用户界面。
- JMeter中负载和性能测试许多不同的服务器类型：网站 - HTTP，HTTPS，SOAP，数据库通过JDBC，LDAP，JMS，邮件 - POP3
- 它是独立于平台的工具。在Linux / UNIX，JMeter中JMeter中shell脚本点击可以调用。在Windows上，它可以调用启动jmeter.bat文件。
- 它具有完整的Swing和轻量级组件支持（预编译的JAR使用包javax.swing中*）。
- JMeter 测试计划存储为XML格式。这意味着可以使用文本编辑器生成一个测试计划。

- 它的完整的多线程框架，允许并发多线程和同步采样不同的功能由单独的线程组采样。
- 它是高度可扩展的。
- 也可用于执行应用程序的自动化测试和功能测试。

# JMeter是如何工作的？

JMeter中模拟一组用户发送到目标服务器的请求和回报目标服务器/应用程序的性能/功能的统计数字表明，通过表格，图形等下图描述了这个过程：



# JMeter环境设置 - JMeter教程

JMeter 是基于 Java 的框架，所以第一个要求是JDK安装在机器上。

## 系统要求

| JDK | 1.6 + |
| --- | --- |
| 内存 | 不限 |
| 硬盘空间 | 不限 |
| 操作系统 | 不限 |

## 第1步 - 验证Java安装在你的机器上

现在，打开控制台并执行以下 java 命令。

| OS | 任务 | 命令 |
| --- | --- | --- |
| Windows | Open Command Console | c:> java -version |
| Linux | Open Command Terminal | $ java -version |
| Mac | Open Terminal | machine:~ joseph$ java -version |

让我们来验证所有的操作系统的输出：

Windows

```
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
```

Linux

```
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
```

Mac

```
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
```

如果没有安装Java，可以从以下网址：http://www.oracle.com/technetwork/java/javase/downloads/index.htmll 下载并安装Java软件开发工具包（SDK）。本教程中使用Java1.7.0_25。

## 第2步：设置JAVA环境

设置 JAVA_HOME 环境变量指向的基本目录的位置，在机器上安装Java。例如;

| OS | 输出 |
| --- | --- |
| Windows | Set the environment variable JAVA_HOME to C:Program FilesJavajdk1.7.0_25 |
| Linux | export JAVA_HOME=/usr/local/java-current |
| Mac | export JAVA_HOME=/Library/Java/Home |

附加 Java编译器的位置到系统路径。

| OS | 输出 |
| --- | --- |
| Windows | Append the string; C:Program FilesJavajdk1.7.0_25in to the end of the system variable, Path. |
| Linux | export PATH=$PATH:$JAVA_HOME/bin/ |
| Mac | not required |

验证Java安装使用命令java-version如上所述。

## 第3步：下载JMeter

下载JMeter最新版本，从 http://jmeter.apache.org/download_jmeter.cgi。在写这篇教程的时候，下载的Apache JMeter2.9，并将它复制到 C:>JMeter 目录中：

目录结构看起来应该像如下：

- apache-jmeter-2.9
- apache-jmeter-2.9in
- apache-jmeter-2.9docs
- apache-jmeter-2.9extras

- apache-jmeter-2.9lib
- apache-jmeter-2.9libext
- apache-jmeter-2.9libjunit
- apache-jmeter-2.9printable_docs

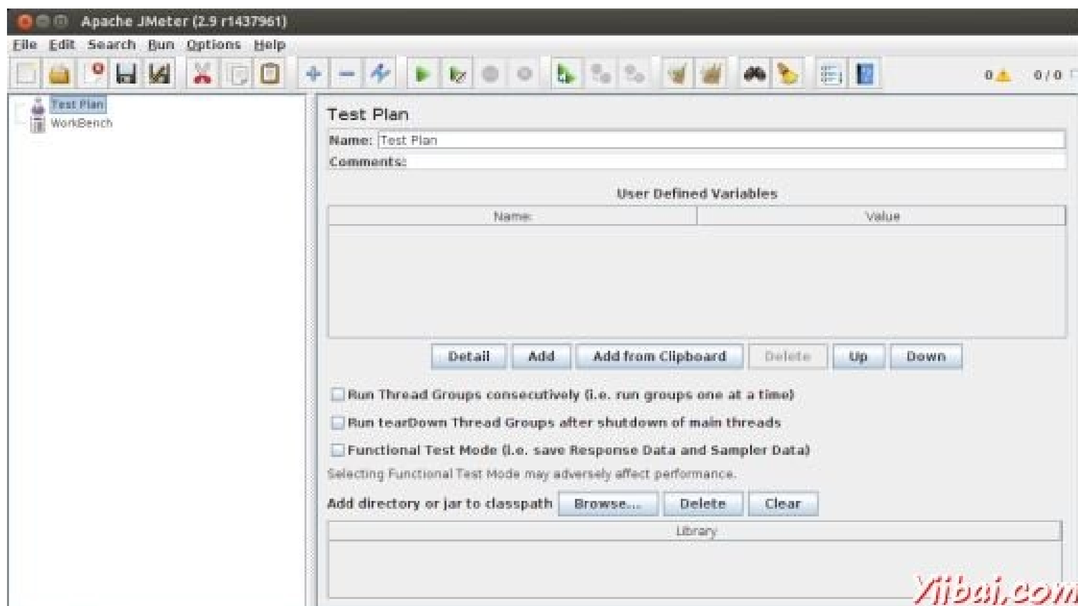
可以重命名的父目录（即Apache的JMeter2.9），但不改变子目录名称。

## 第4步：运行JMeter

一旦下载JMeter，跳转到bin目录。在我们本教程中，这将是 /home/manisha/apache-jmeter-2.9/bin。现在点击以下：

| OS | 输出 |
|---|---|
| Windows | jmeter.bat |
| Linux | jmeter.sh |
| Mac | jmeter.sh |

JMeter的GUI点击上述文件后，经过短暂的停顿，应该会出现在下面的图片中看到，这是一个Swing应用程序：




这是主页面，默认页面的工具。

# JMeter创建测试计划 - JMeter教程

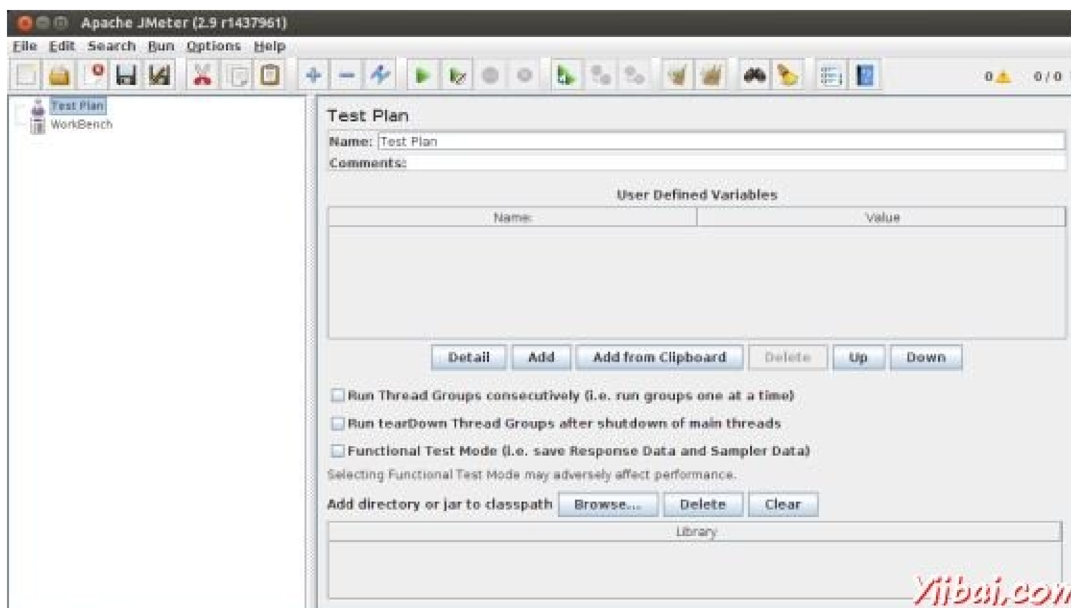
## 测试计划是什么？

测试计划定义如何和测试，并提供了一个布局。例如，Web应用程序以及客户端服务器应用程序。它可以被看作作为容器运行测试。一个完整的测试计划将包括一个或多个元素，如线程组，逻辑控制器，样品产生控制器，监听器，定时器，断言和配置元素。测试计划必须至少有一个线程组。在下一章中，我们将讨论这些元素的细节 测试计划元素.

按照下面的步骤来写一个测试计划：

### 启动JMeter窗口

打开JMeter窗口通过点击 /home/manisha/apache-jmeter-2.9/bin/jmeter.sh。JMeter窗口会出现如下图：



这个JMeter窗口什么都还没有添加。上述窗口的详细信息如下：

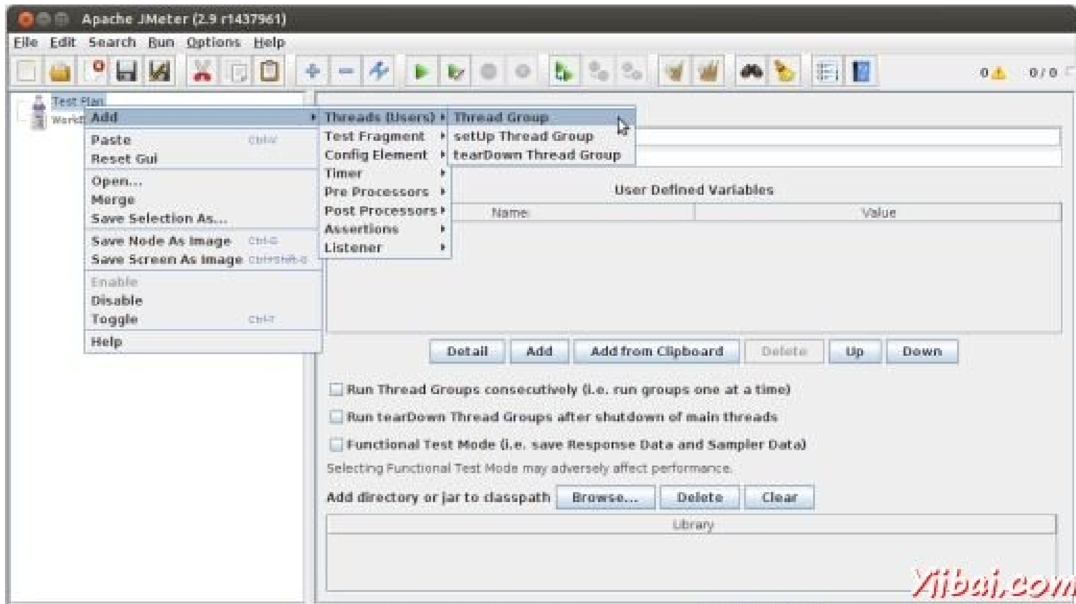
- 保持真正的测试计划，测试计划节点。
- 工作台节点只是提供了一个地方暂时存放在不使用时，测试元素复制/粘贴的目的。当您保存测试计划，工作台项目将不保存它。

### 添加/删除元素

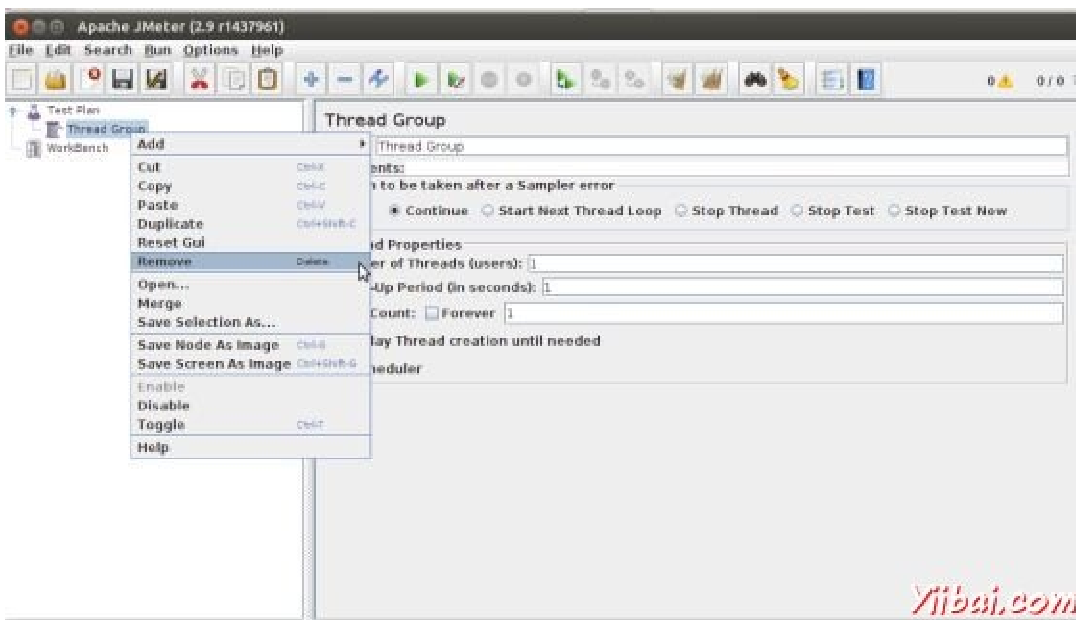
通过右键点击测试计划节点，并从“add”列表中选择一个新的元素，元素（将在下一章测试计划要素讨论），可以添加一个测试计划。

另外，元素可以从文件中加载，并通过选择“merge”或“open”选项添加。

例如，让我们添加一个线程组元件测试计划如下所示：

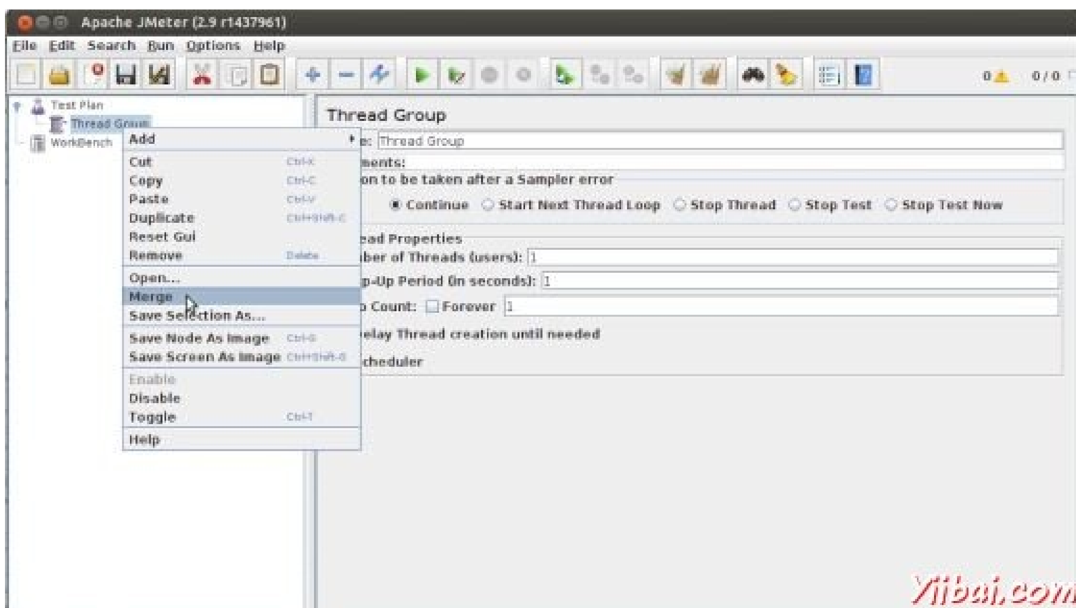



要删除一个元素，确保元素被选中，右键单击该元素，然后选择“remove”选项。




## 加载和保存元素

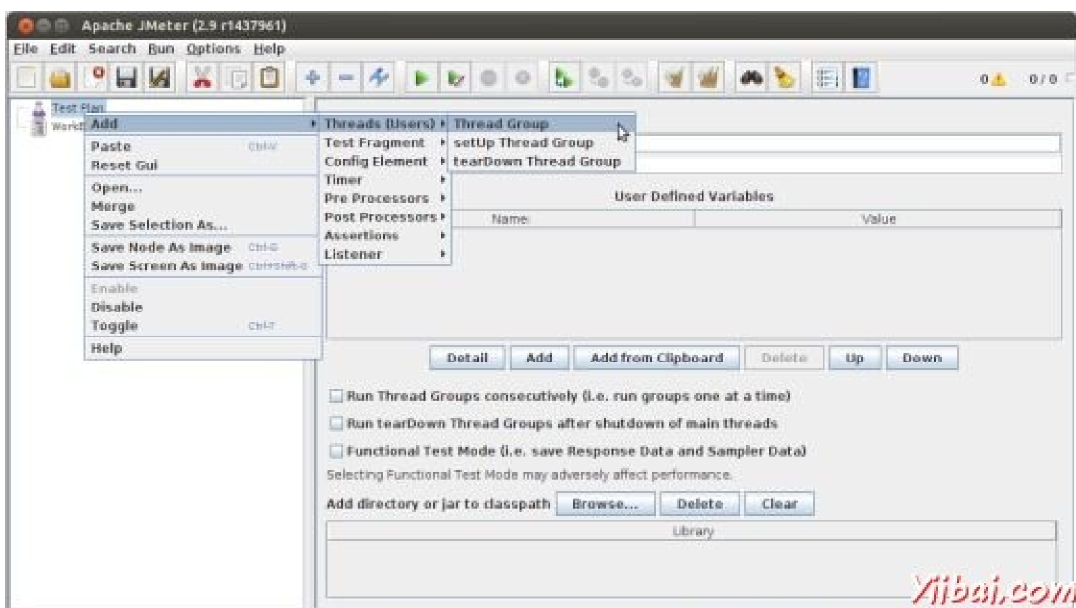
从文件加载一个元素，右键单击您要添加的加载元件对现有树元素，并选择“merge”选项。选择文件保存元素。 JMeter会合并的元素，放到树上。




为了保存树元素，元素上点击右键并选择选择另存为...选项。 JMeter会保存选定的元素，再加上它下面的所有子元素。默认情况下，不保存JMeter 的元素，需要明确地保存它，如前面提到的。

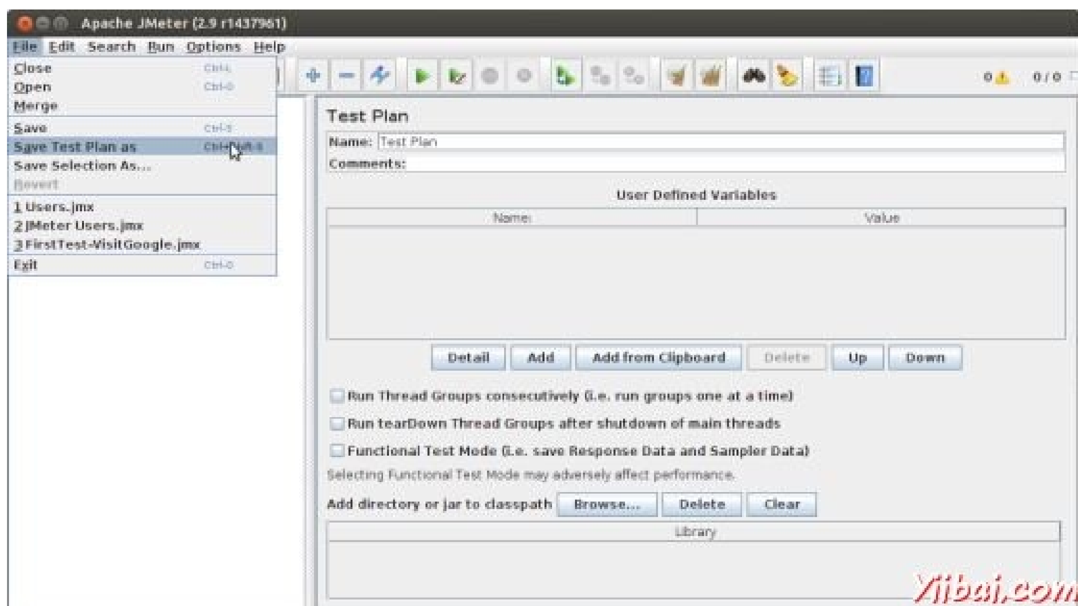
## 配置树元素

目前在JMeter 右手帧的控制，可配置的测试计划中的任何元素。这些控件允许配置特定的测试元件的行为。例如线程组可配置的用户数量上升期等如下：



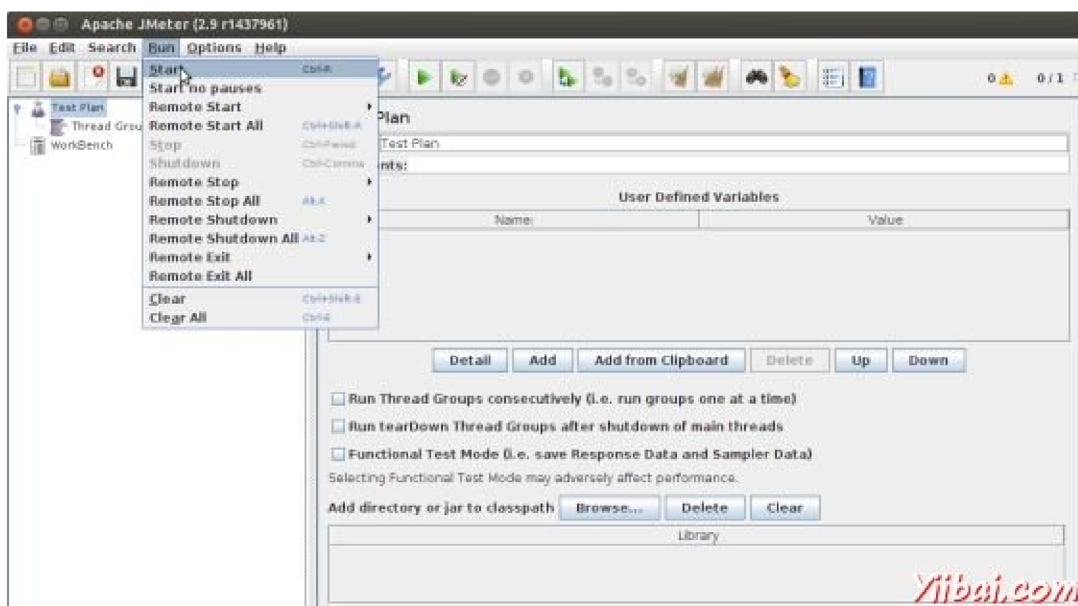

## 保存测试计划

您可以保存整个测试计划，可以通过使用“Save ”或“Save Test Plan As ....”从“文件”菜单。




## 运行一个测试计划

您可以运行您的测试计划从Run菜单项中选择“Start ”（控制+ R）。当运行JMeter是，它显示了一个绿色的小盒子，右手端的部分，只是在菜单栏下。




左侧的绿色方块的数字是活动线程/线程总数的数量。这些只适用于本地运行的测试，他们不包括任何远程系统上使用客户 - 服务器模式时启动的线程。

## 停止测试

停止测试方法有两种：

- 使用stop（Control + '.'）。这立即停止线程如果可能的话。
- 使用shutdown（Control + ','）。这就要求线程停止在任何当前工作的结束。

# JMeter数据库测试计划 - JMeter教程

在本章中，我们将看到如何创建一个简单的测试计划，测试数据库服务器。对于我们的测试目的，我们使用MySQL数据库服务器。您可以使用任何其他数据库进行测试。MYSQ的安装和创建表，请参阅 MYSQL教程。

安装MySQL以后，请按照以下步骤设置数据库：

- 创建一个数据库名称 "tutorial".
- 创建一个表 tutorials_tbl.
- 插入记录到 tutorials_tbl :

```
mysql&gt; use TUTORIALS;
Database changed
mysql&gt; INSERT INTO tutorials_tbl
     -&gt;(tutorial_title, tutorial_author, submission_date)
     -&gt;VALUES
     -&gt;("Learn PHP", "John Poul", NOW());
Query OK, 1 row affected (0.01 sec)
mysql&gt; INSERT INTO tutorials_tbl
     -&gt;(tutorial_title, tutorial_author, submission_date)
     -&gt;VALUES
     -&gt;("Learn MySQL", "Abdul S", NOW());
Query OK, 1 row affected (0.01 sec)
mysql&gt; INSERT INTO tutorials_tbl
     -&gt;(tutorial_title, tutorial_author, submission_date)
     -&gt;VALUES
     -&gt;("JAVA Tutorial", "Sanjay", '2007-05-06');
Query OK, 1 row affected (0.01 sec)
mysql&gt;
```

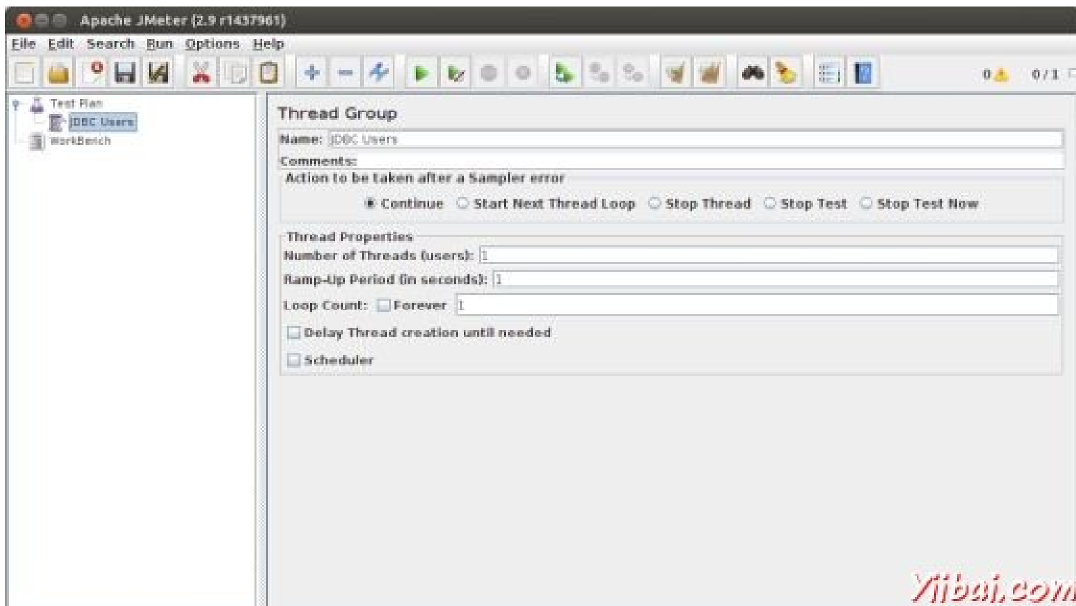
- 复制JDBC驱动程序到 /home/manisha/apache-jmeter-2.9/lib.

## 创建JMeter测试计划

首先，让我们启动JMeter /home/manisha/apache-jmeter-2.9/bin/jmeter.sh.

### 添加用户

现在，创建一个线程组，右键点击 Test Plan > Add> Threads(Users)> Thread Group. 根据测试计划节点将添加线程组。重命名此线程为JDBC用户。




我们不会改变线程组的默认属性。

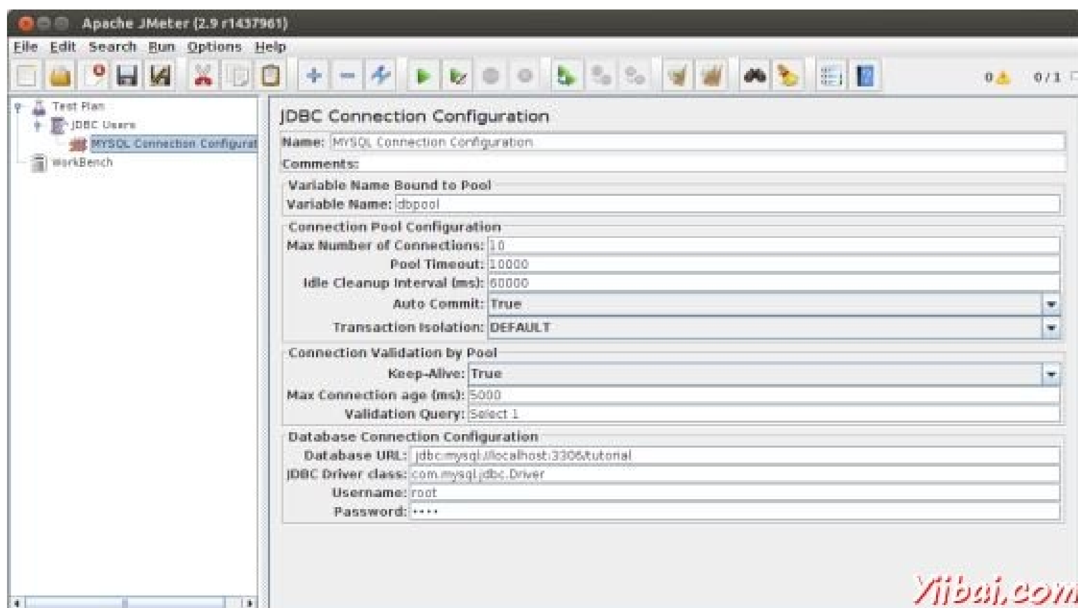
## 添加JDBC请求

现在，我们已经定义了我们的用户，它是时间来定义，他们将要执行的任务。在本节中将指定JDBC请求执行。 JDBC Users元件上右击，选择 Add > Config Element > JDBC Connection Configuration.

设置以下字段（我们使用的是MySQL数据库教程）：

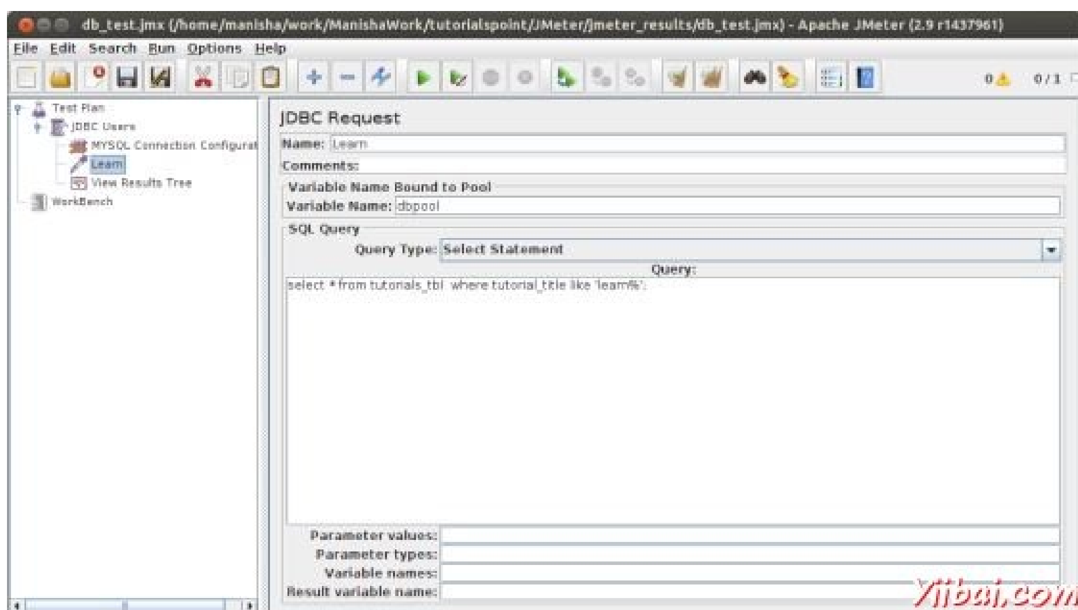
- 变量名绑定到池。这需要唯一地标识该配置。它是用来由JDBC采样器，以确定要使用的配置。作为测试，我们把它命名为 test
- Database URL: jdbc:mysql://localhost:3306/tutorial
- JDBC Driver class: com.mysql.jdbc.Driver
- 用户名: root
- 密码: root的密码

在屏幕上的其他领域，可以留为默认值，如下所示：




添加一个JDBC请求是指上面定义的JDBC配置池。选择JDBC Users元件，单击鼠标右键得到添加菜单，然后选择 Add > Sampler > JDBC Request. 然后，选择这个新的元素，以查看它的控制面板。编辑属性如下：

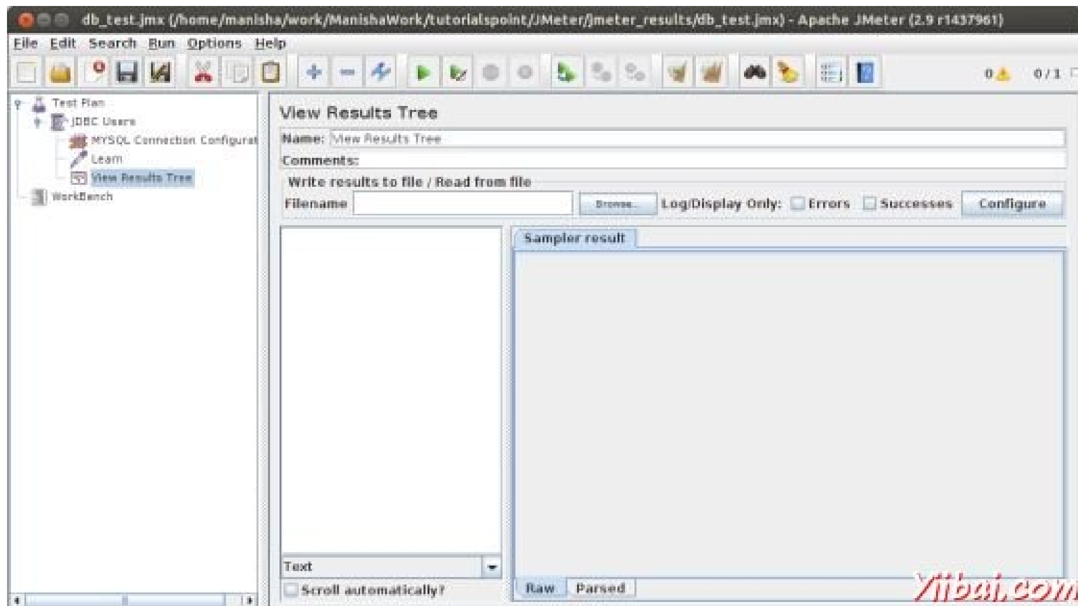
- 变量名绑定到池。这需要唯一地标识该配置。它是用来由JDBC采样器，以确定要使用的配置。我们将其命名为 test
- Name: Learn
- Enter the Pool Name: test (same as in the configuration element)
- Query Type: Select statement
- Enter the SQL Query String field.




## 创建侦听器

现在添加Listener元素。此元素负责存储所有JDBC请求的结果，在一个文件中，并呈现出可视化的数据模型。

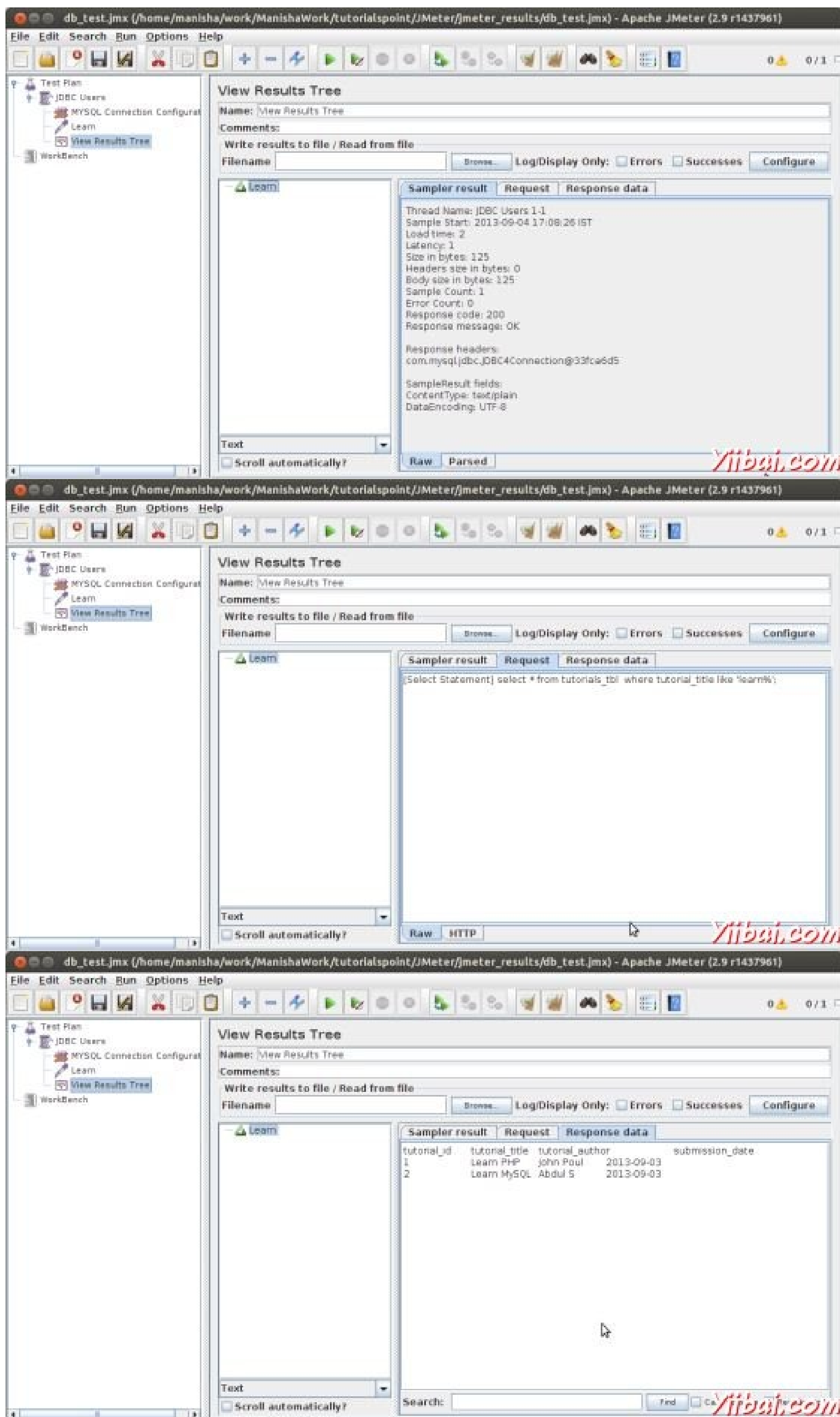
选择JDBC Users元件，并添加一个查看结果树监听器(Add > Listener > View Results Tree).




## 保存并执行测试计划

现在保存的以上测试计划db_test.jmx。执行本测试计划使用 Run > Start 选项.

## 校验输出



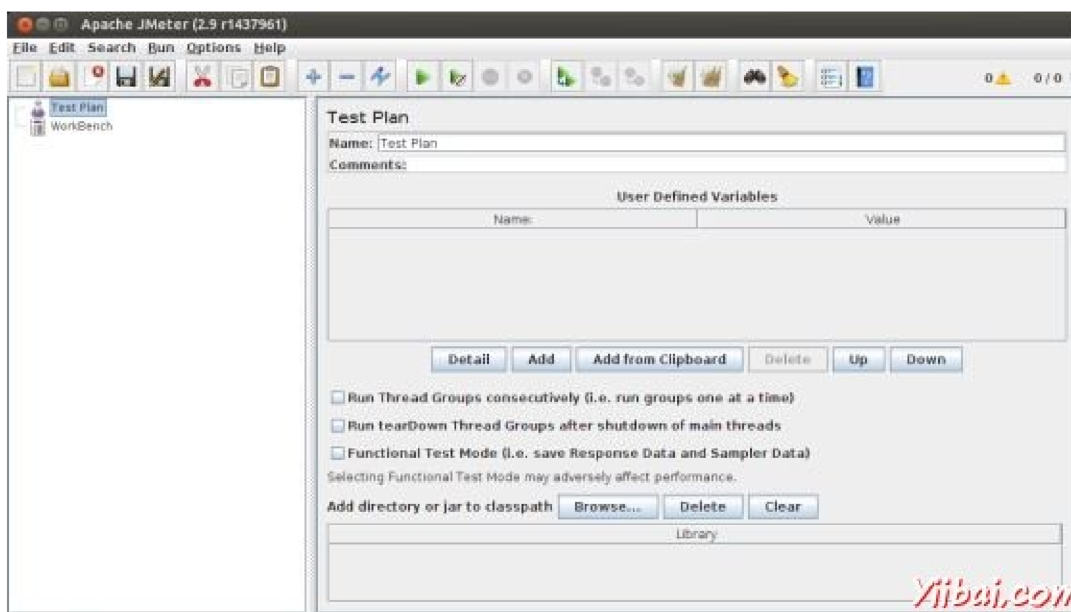
在最后图像，可以看到，2条记录被选择。

# JMeter Web测试计划 - JMeter教程

让我们建立一个简单的测试计划，测试一个网页。我们将编写一个测试计划，使我们可以在Apache JMeter的性能测试一个网页所示页面网址：http://www.yiibai.com/

## 启动JMeter

打开JMeter窗口通过点击 /home/manisha/apache-jmeter-2.9/bin/jmeter.sh。JMeter窗口会出现如下图：

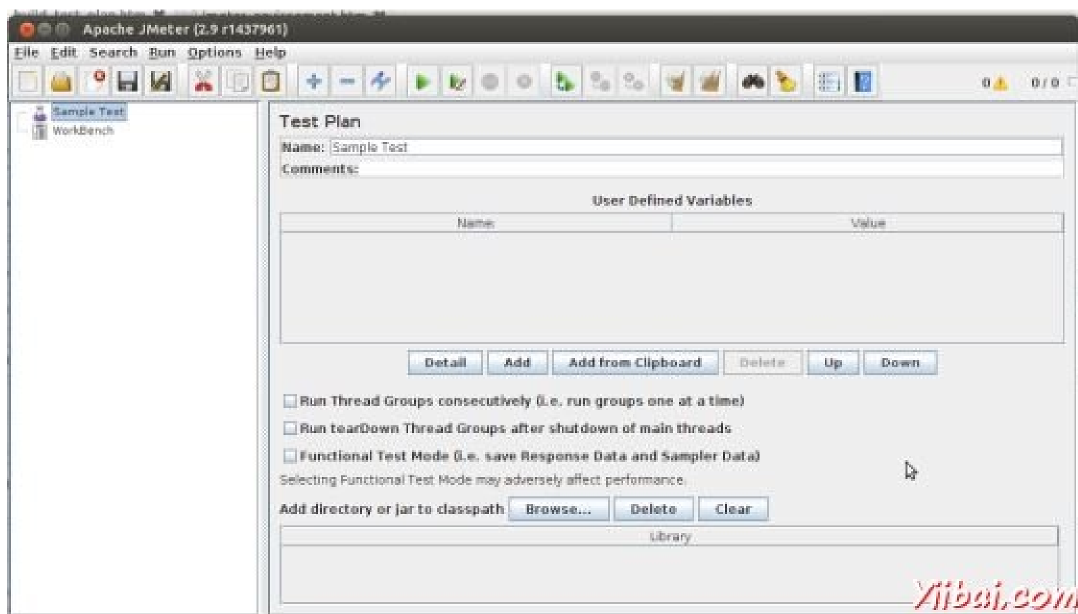



这是一个JMeter窗口什么都尚未添加。上述窗口的详细信息如下：

- 保持真正的测试计划，测试计划节点。
- 保持工作台节点是临时的东西。

## 重命名测试计划

更改测试计划节点的名称，在命名文本框中样品测试。必须改变焦点工作台节点和后台，测试计划节点看名字反映。

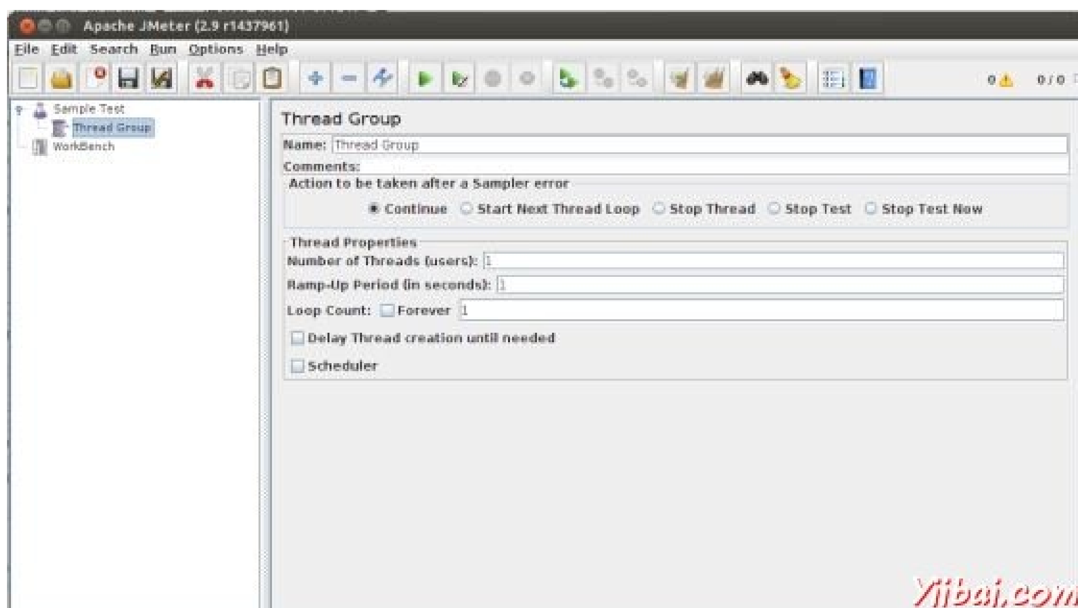



## 添加线程组

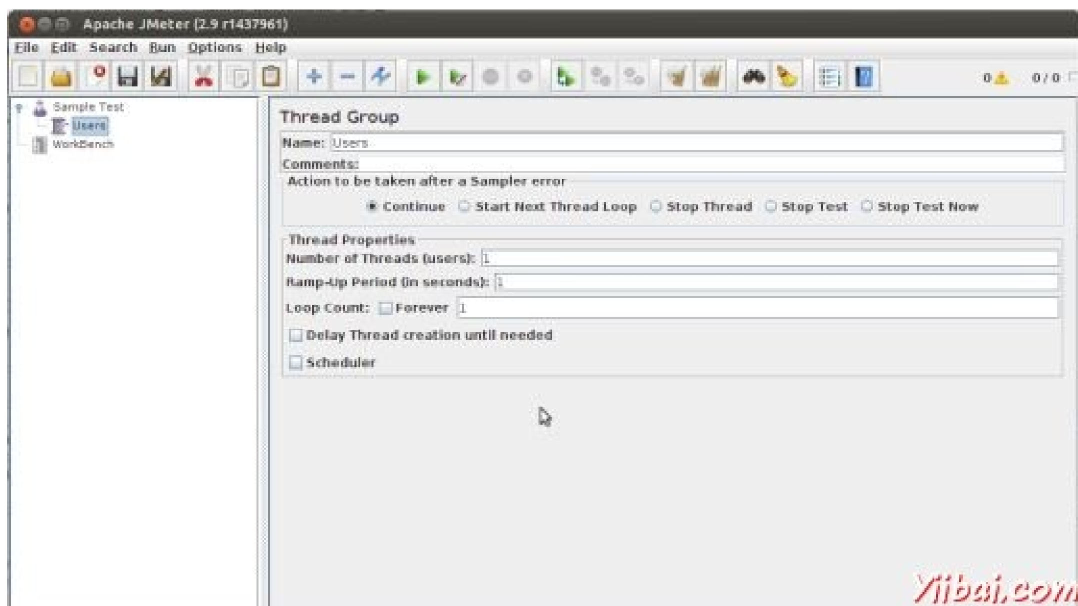
现在，我们将增加在窗口中的第一个元素。我们将添加一个线程组，这是所有其他元素，如取样器，控制器，监听的占位符。可以配置要模拟的用户数。

在JMeter中所有节点的元素被添加使用上下文菜单。想增加一个子元素节点，右击该元素。选择合适的选项添加。

右键单击 Sample Test(our Test Plan )> Add> Threads(Users)> Thread Group. 将添加线程组，根据测试计划（样品测试）节点。

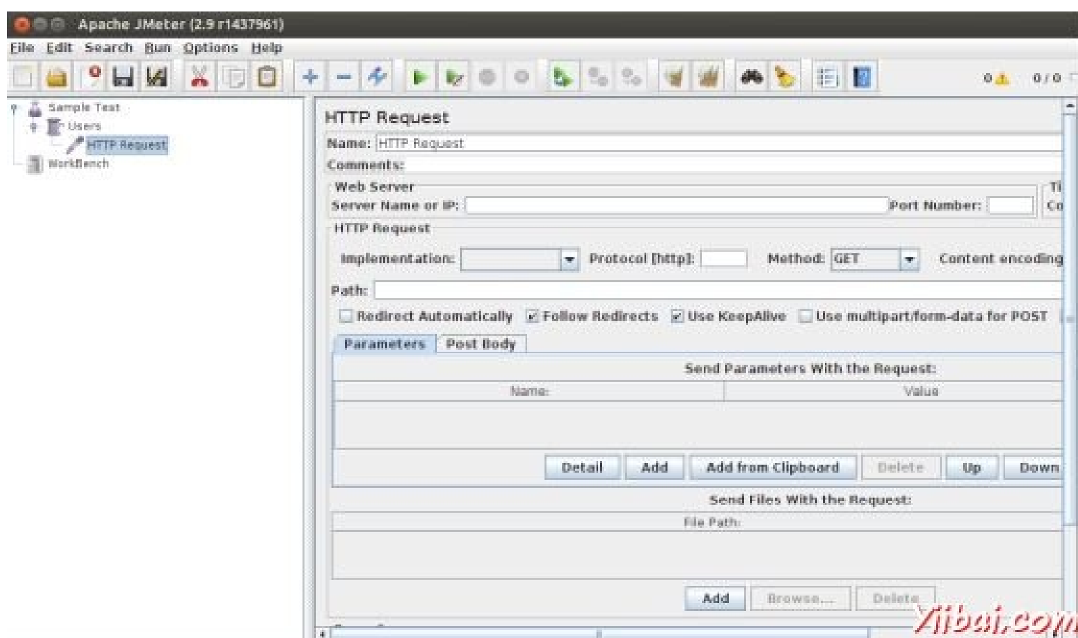



作为用户，我们将其命名线程组。对于我们这个元素是指用户访问的TutorialsPoint主页。

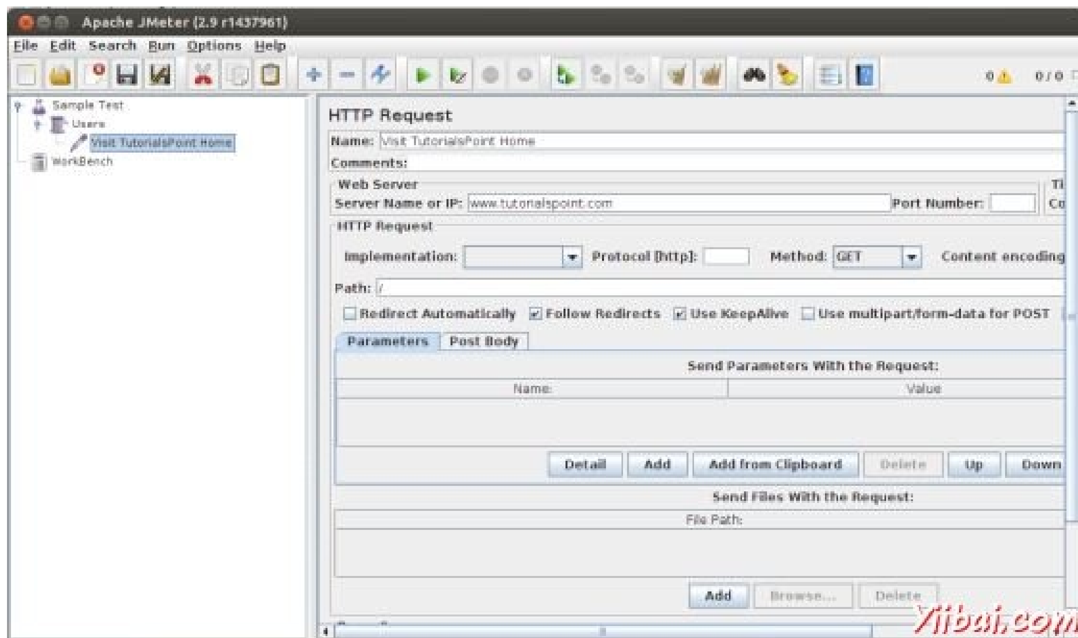



## ADD采样器

现在，我们必须在我们的线程组（用户）添加一个采样。添加线程组做得比较早，这一次我们将开启的线程组（用户）节点的上下文菜单中通过右击并选择 Add > Sampler> HTTP请求选项，我们将添加HTTP请求采样。



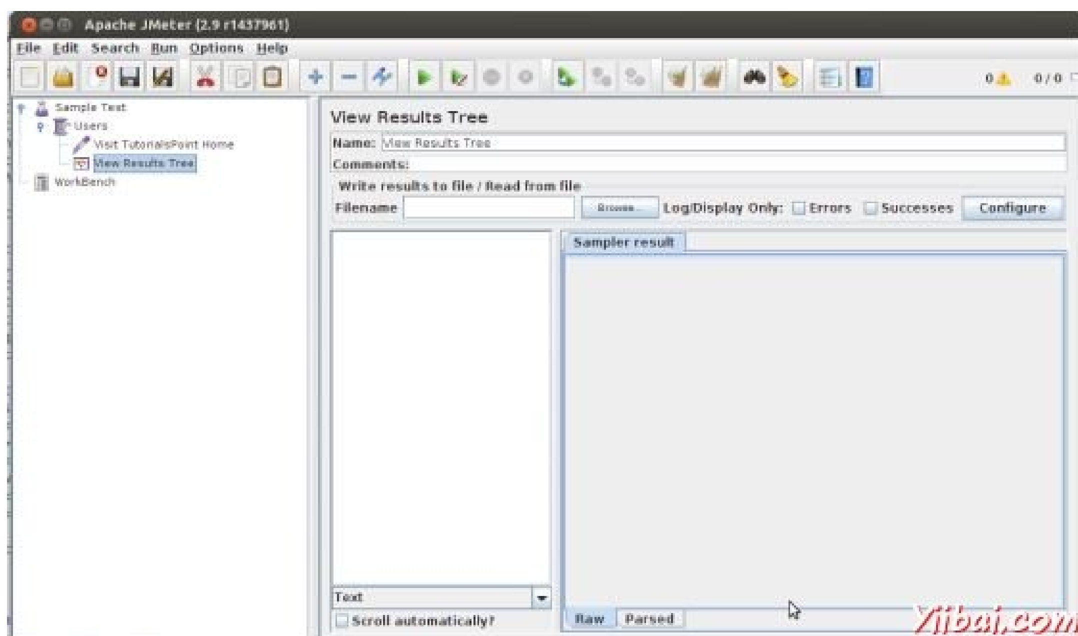

这将添加一个空的HTTP请求采样线程组（用户）节点下。让我们配置此节点元素：




- 名称：我们将改变名称，以反映我们要实现的动作是什么。我们将它命名为访问TutorialsPoint首页
- 服务器名称或IP：在这里，我们必须键入Web服务器的名称。在我们的例子中，它是www.yiibai.com。（包含http://部分不被写入，这仅仅是的服务器的名称或IP）
- 协议：保持空白，这里默认使用HTTP协议。
- 路径：键入路径为/（斜线）。这意味着访问服务器的根页。

## 添加侦听器

我们将添加一个侦听器。让我们添加查看结果树监听线程组（用户）节点下。这将确保采样的结果将可查看该监听节点元素。打开上下文菜单上点击右键选择Add > Listener > View Results Tree选项添加监听线程组（用户）。

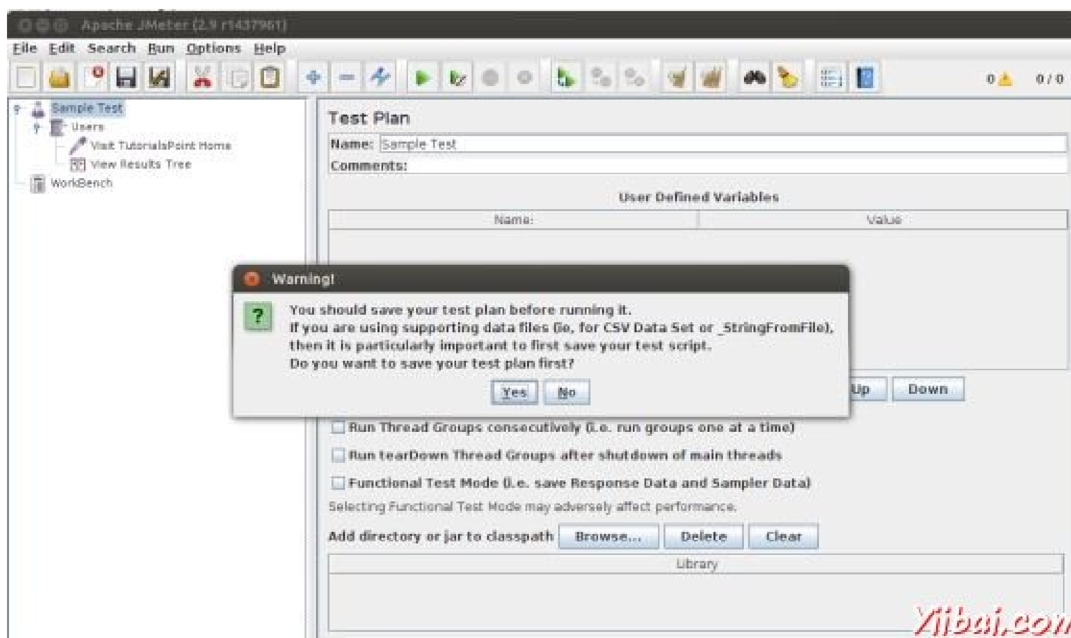


## 运行此测试计划

现在所有的设置，让我们来执行测试计划。随着线程组的配置（用户），我们保留了它所有默认值。这意味着JMeter的执行采样一次。这将是像一个单一的用户只有一次。

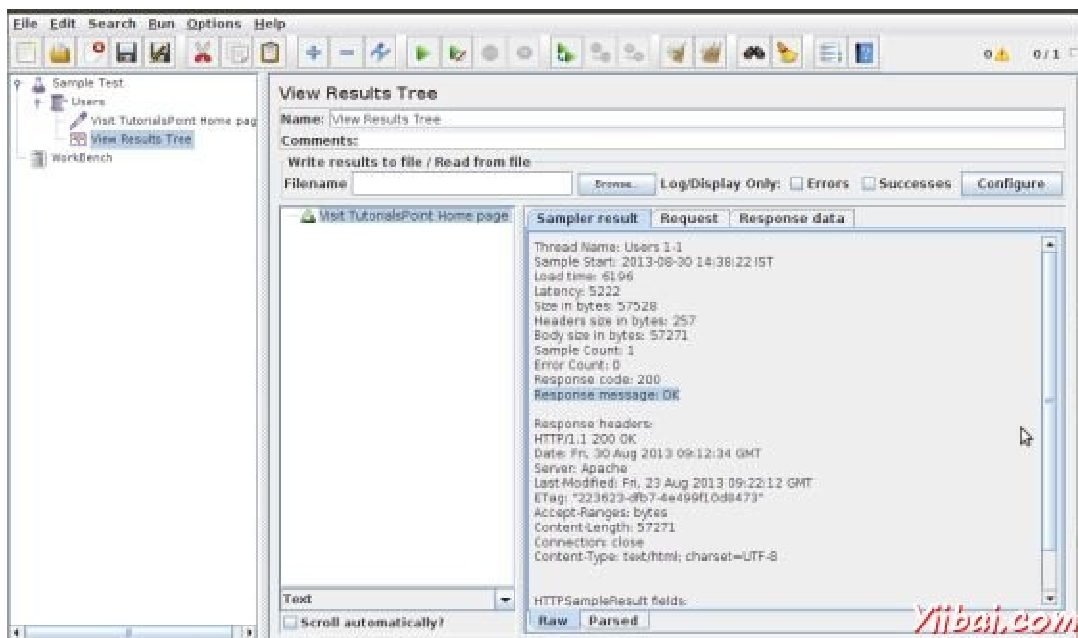
这是类似的，喜欢的用户通过浏览器访问的网页，只有在这里，我们正在通过JMeter的采样。使用 Run > Start 选项，我们将执行测试计划。

Apache JMeter的的要求我们在实际测试开始之前，在磁盘文件保存测试计划。这是很重要的，如果我们要运行的测试计划，一遍又一遍。如果说通过单击“No ”选项，不保存，它将运行而不进行保存。

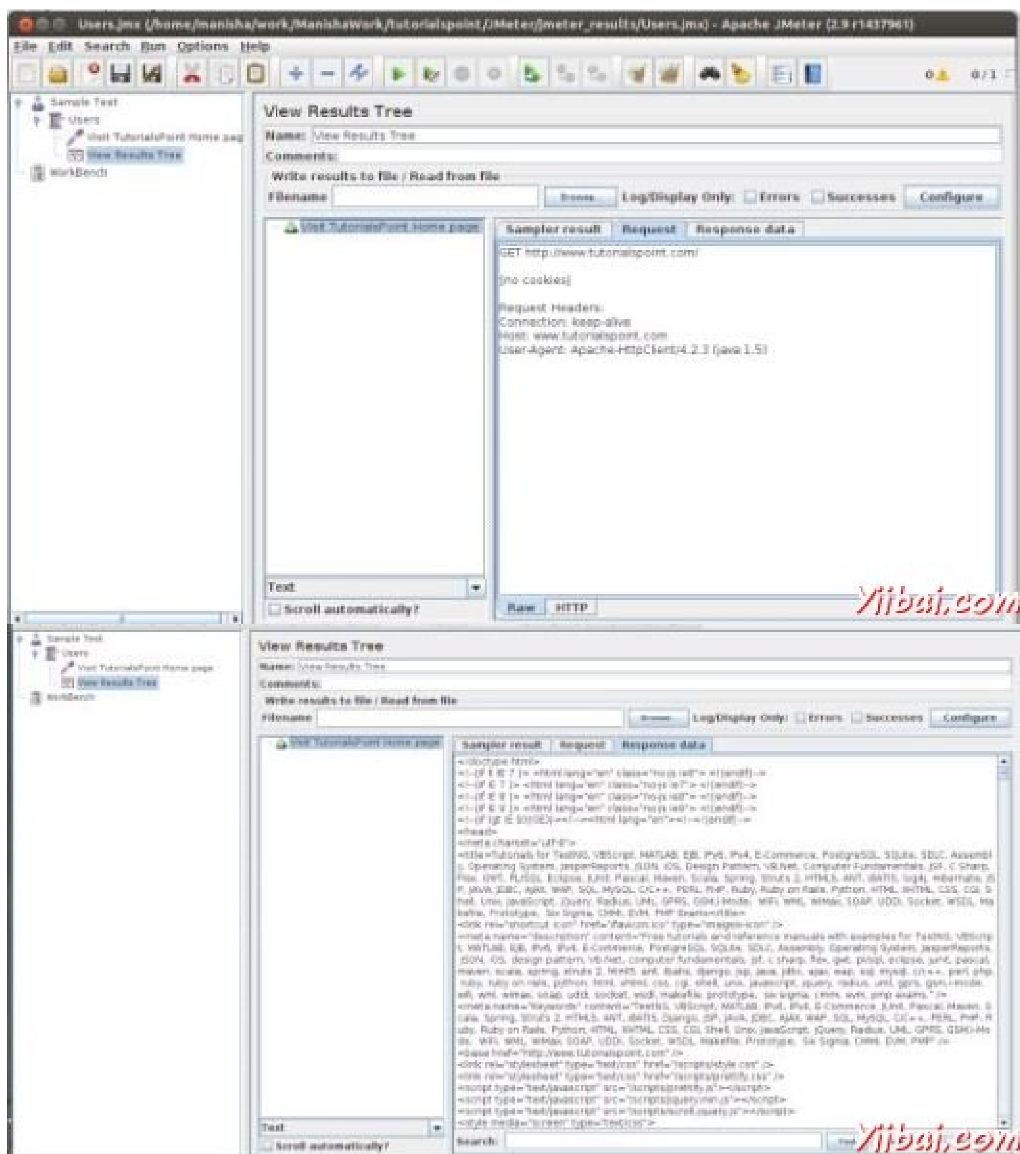



## 查看输出

我们已经保持了设置的线程组作为单个线程（这意味着只有一个用户）和循环时间（这意味着只运行一次），因此，我们将得到单一交易中查看结果树监听的结果。



上述结果的详细信息如下：



- 针对名称访问TutorialsPoint首页绿色表示成功。
- JMeter已存储了所有的头和由Web服务器发送响应，并准备向我们展示了在许多方面。
- 第一个标签是采样结果。它显示的JMeter的数据以及由Web服务器返回的数据。
- 第二个选项卡的请求，其中示出所有的数据作为请求的一部分被发送到Web服务器。
- 最后一个标签是响应数据。听者在此选项卡中显示了从服务器接收到的数据，因为它是在文本格式。

这仅仅是一个简单的测试计划，执行只有一个请求。但JMeter真实实力，像很多用户都发送发送相同的请求。来测试Web服务器与多个用户，我们将不得不更改线程组（用户）设置。

# JMeter数据库测试计划 - JMeter教程

在本章中，我们将看到如何创建一个简单的测试计划，测试数据库服务器。对于我们的测试目的，我们使用MySQL数据库服务器。您可以使用任何其他数据库进行测试。MYSQ的安装和创建表，请参阅 MYSQL教程。

安装MySQL以后，请按照以下步骤设置数据库：

- 创建一个数据库名称 "tutorial".
- 创建一个表 tutorials_tbl.
- 插入记录到 tutorials_tbl :

```
mysql&gt; use TUTORIALS;
Database changed
mysql&gt; INSERT INTO tutorials_tbl
     -&gt;(tutorial_title, tutorial_author, submission_date)
     -&gt;VALUES
     -&gt;("Learn PHP", "John Poul", NOW());
Query OK, 1 row affected (0.01 sec)
mysql&gt; INSERT INTO tutorials_tbl
     -&gt;(tutorial_title, tutorial_author, submission_date)
     -&gt;VALUES
     -&gt;("Learn MySQL", "Abdul S", NOW());
Query OK, 1 row affected (0.01 sec)
mysql&gt; INSERT INTO tutorials_tbl
     -&gt;(tutorial_title, tutorial_author, submission_date)
     -&gt;VALUES
     -&gt;("JAVA Tutorial", "Sanjay", '2007-05-06');
Query OK, 1 row affected (0.01 sec)
mysql&gt;
```

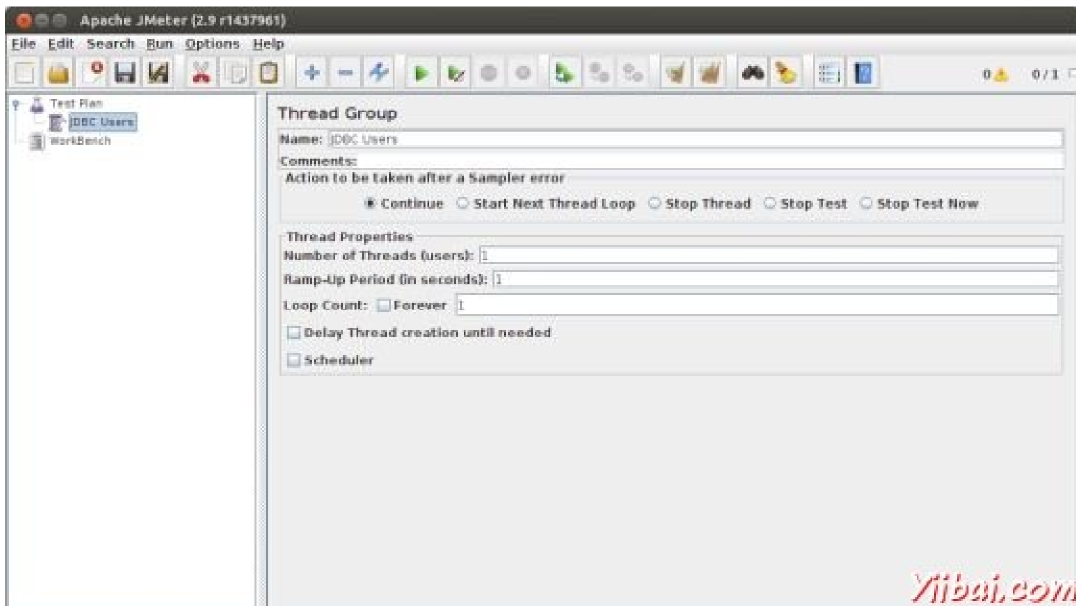
- 复制JDBC驱动程序到 /home/manisha/apache-jmeter-2.9/lib.

## 创建JMeter测试计划

首先，让我们启动JMeter /home/manisha/apache-jmeter-2.9/bin/jmeter.sh.

### 添加用户

现在，创建一个线程组，右键点击 Test Plan > Add> Threads(Users)> Thread Group. 根据测试计划节点将添加线程组。重命名此线程为JDBC用户。




我们不会改变线程组的默认属性。

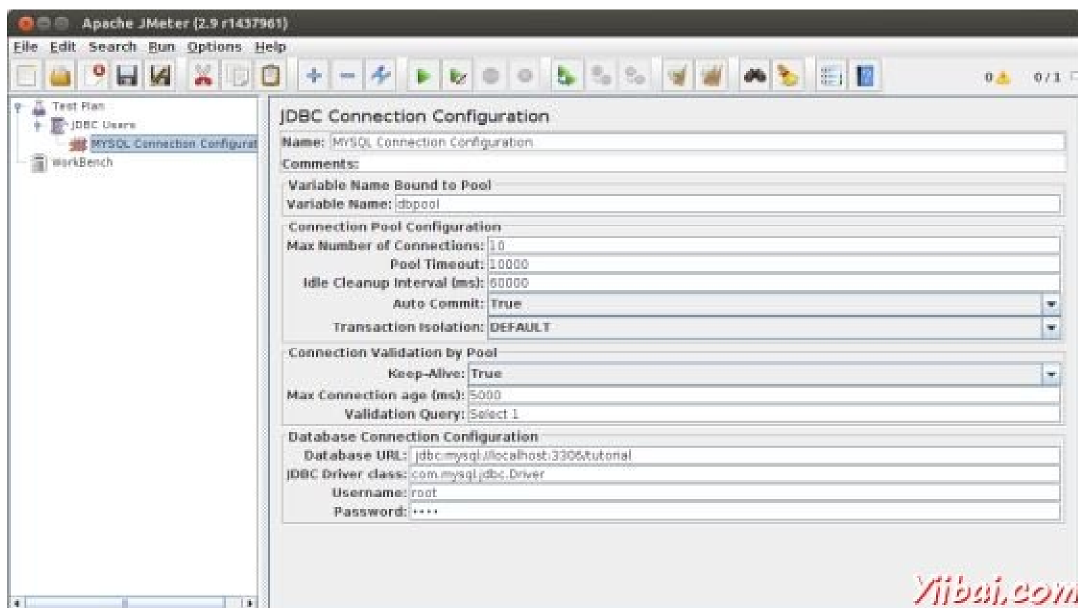
## 添加JDBC请求

现在，我们已经定义了我们的用户，它是时间来定义，他们将要执行的任务。在本节中将指定JDBC请求执行。 JDBC Users元件上右击，选择 Add > Config Element > JDBC Connection Configuration.

设置以下字段（我们使用的是MySQL数据库教程）：

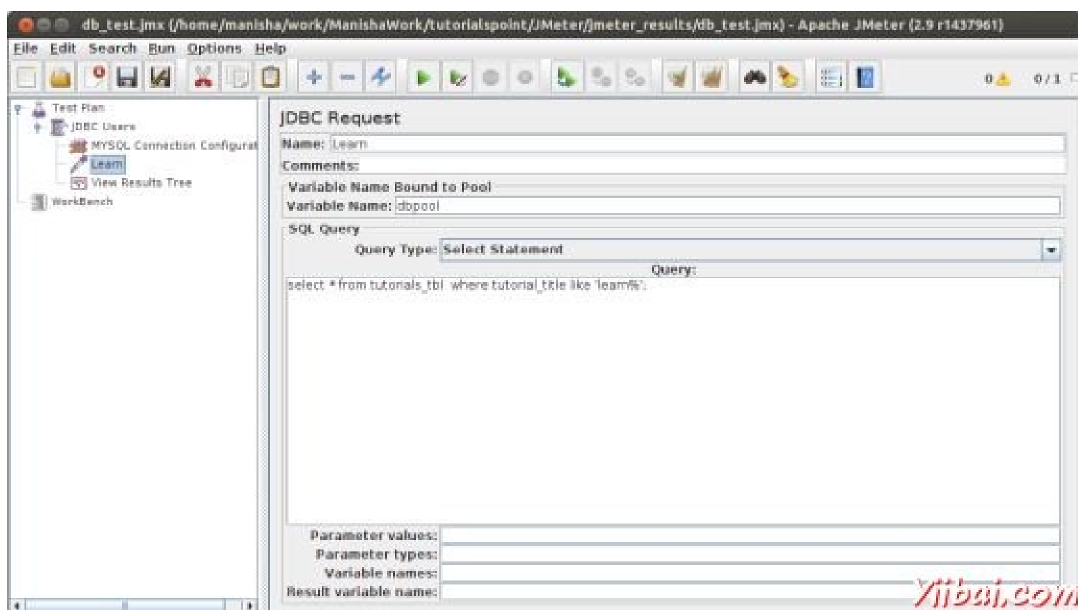
- 变量名绑定到池。这需要唯一地标识该配置。它是用来由JDBC采样器，以确定要使用的配置。作为测试，我们把它命名为 test
- Database URL: jdbc:mysql://localhost:3306/tutorial
- JDBC Driver class: com.mysql.jdbc.Driver
- 用户名: root
- 密码: root的密码

在屏幕上的其他领域，可以留为默认值，如下所示：




添加一个JDBC请求是指上面定义的JDBC配置池。选择JDBC Users元件，单击鼠标右键得到添加菜单，然后选择 Add > Sampler > JDBC Request. 然后，选择这个新的元素，以查看它的控制面板。编辑属性如下：

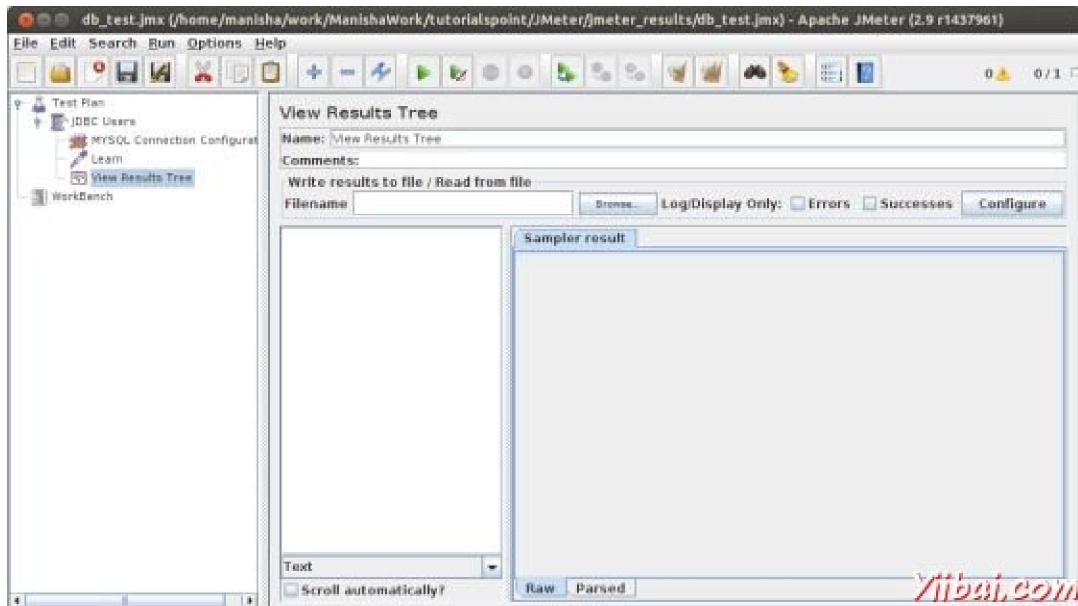
- 变量名绑定到池。这需要唯一地标识该配置。它是用来由JDBC采样器，以确定要使用的配置。我们将其命名为 test
- Name: Learn
- Enter the Pool Name: test (same as in the configuration element)
- Query Type: Select statement
- Enter the SQL Query String field.




## 创建侦听器

现在添加Listener元素。此元素负责存储所有JDBC请求的结果，在一个文件中，并呈现出可视化的数据模型。

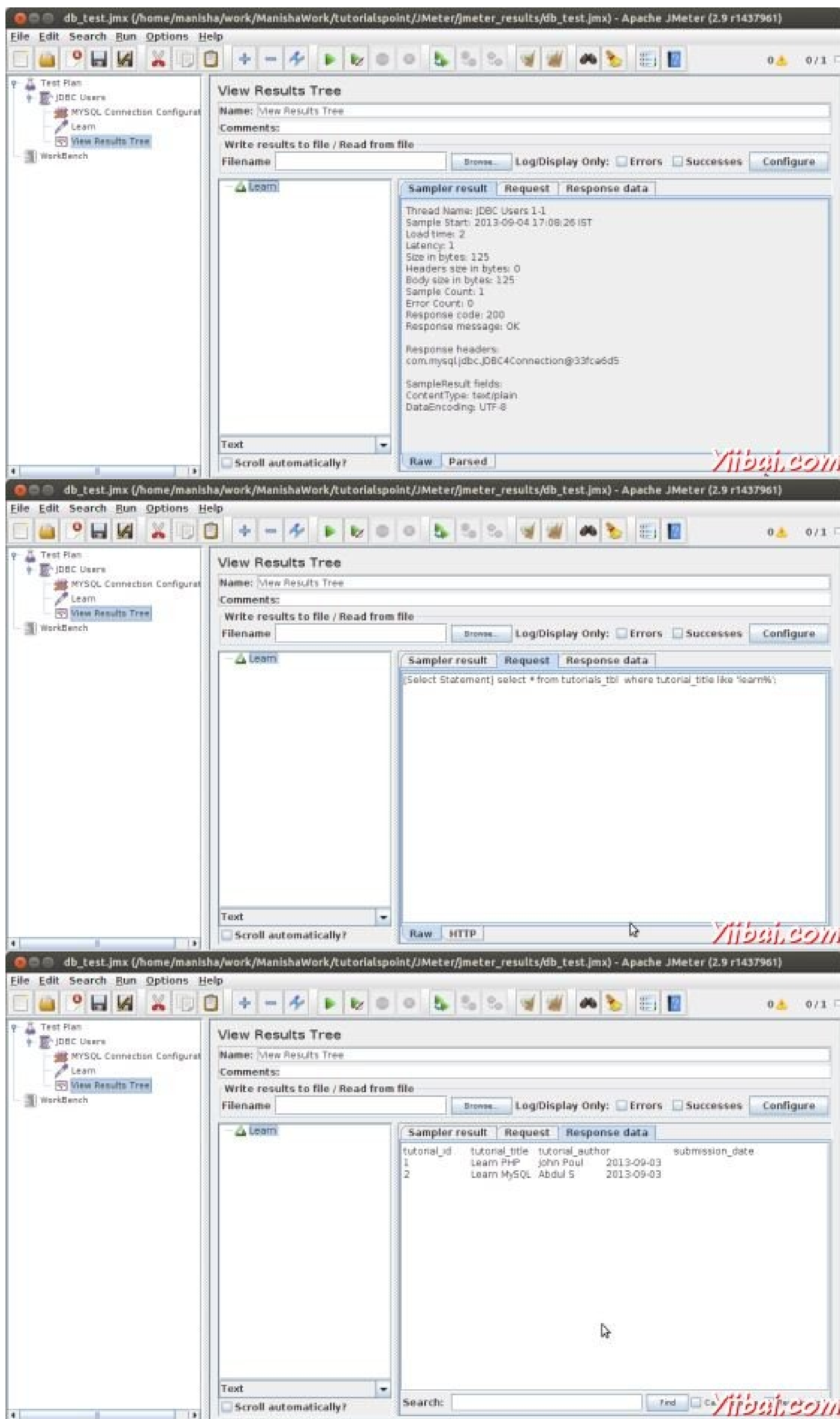
选择JDBC Users元件，并添加一个查看结果树监听器(Add > Listener > View Results Tree).



## 保存并执行测试计划

现在保存的以上测试计划db_test.jmx。执行本测试计划使用 Run > Start 选项.

## 校验输出



在最后图像，可以看到，2条记录被选择。

# jMeter FTP测试计划 - JMeter教程

在本章中，我们将看到如何使用JMeter 测试FTP站点。让我们创建一个测试计划，测试FTP站点。

## 重命名测试计划

通过点击启动 JMeter 窗口 **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh.** 点击测试计划节点上。重命名此测试计划节点 **TestFTPSite**.

## 添加线程组

添加一个线程组，这是所有其他元素，如取样器，控制器，监听的占位符。右键单击 **TestFTPSite(our Test Plan) > Add > Threads(Users) > Thread Group.** 线程组将添加根据测试计划（**TestFTPSite**）的节点。

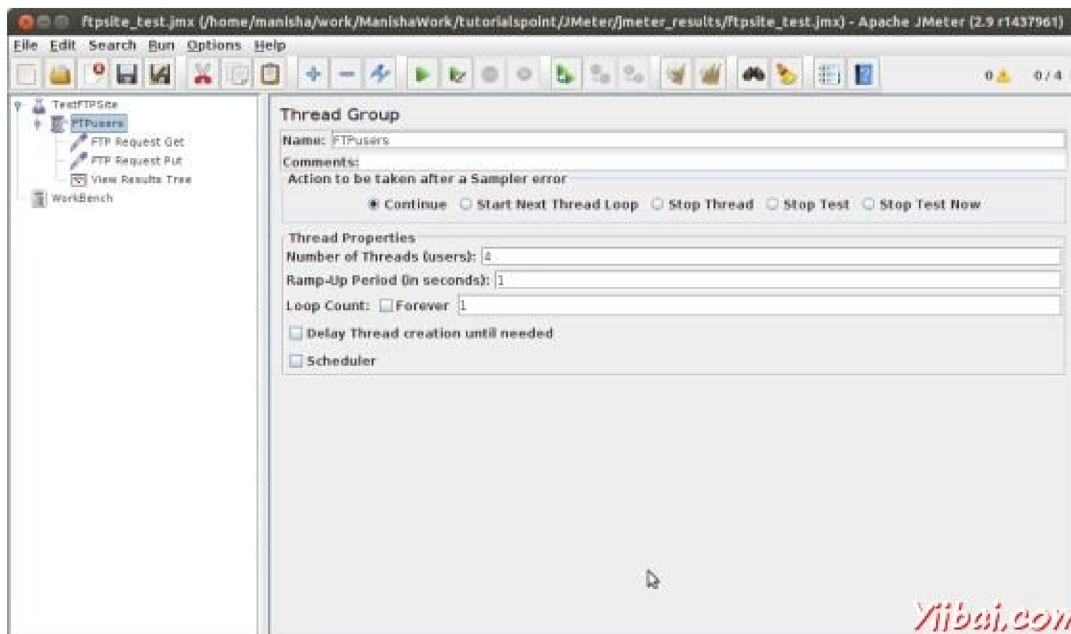
接下来，让我们修改线程组的默认属性，以满足我们的测试。改变以下属性：

Name: FTPusers

线程数（用户）: 4

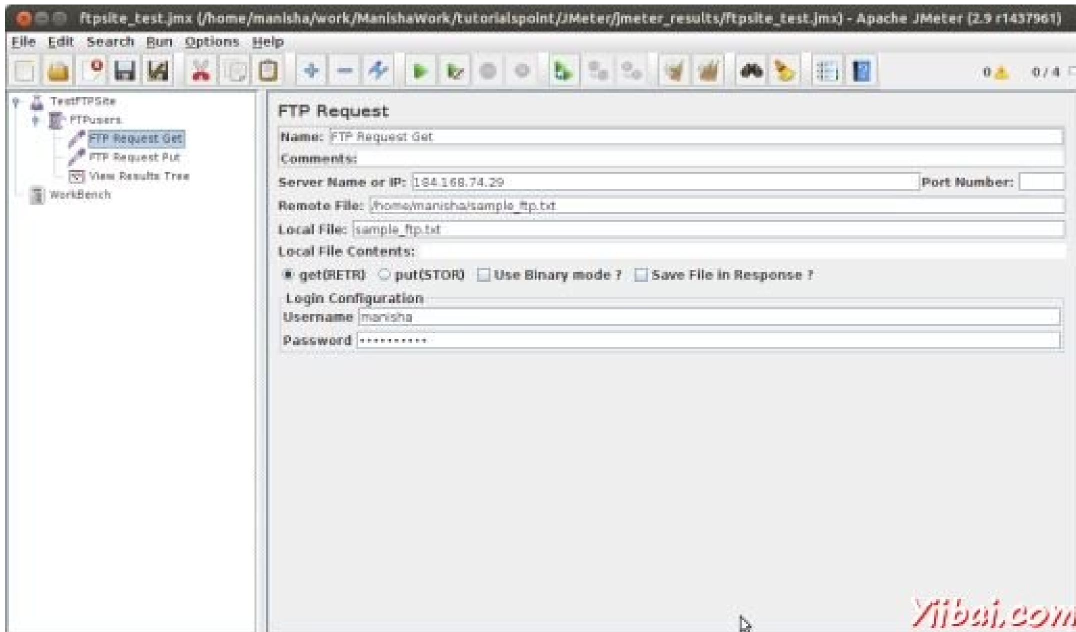
Ramp-Up Period: 离开默认值0秒。

Loop Count:1




## 添加采样器的FTP请求

现在，我们已经定义了我们的用户，它是时间来定义，他们将要执行的任务。我们将添加FTP请求元素。我们将添加两个FTP请求元素，将检索一个文件，将FTP站点上的文件。开始由选择FTPUSERS元素。点击鼠标右键得到添加菜单，然后选择 **Add > Sampler > FTP Request.** 然后，选择FTP请求树中的元素，并在下面的图片编辑下列属性：



这个元素中输入下列详细信息：

Name: FTP Request Get

Server Name or IP: 184.118.14.9

Remote File: /home/manisha/sample_ftp.txt

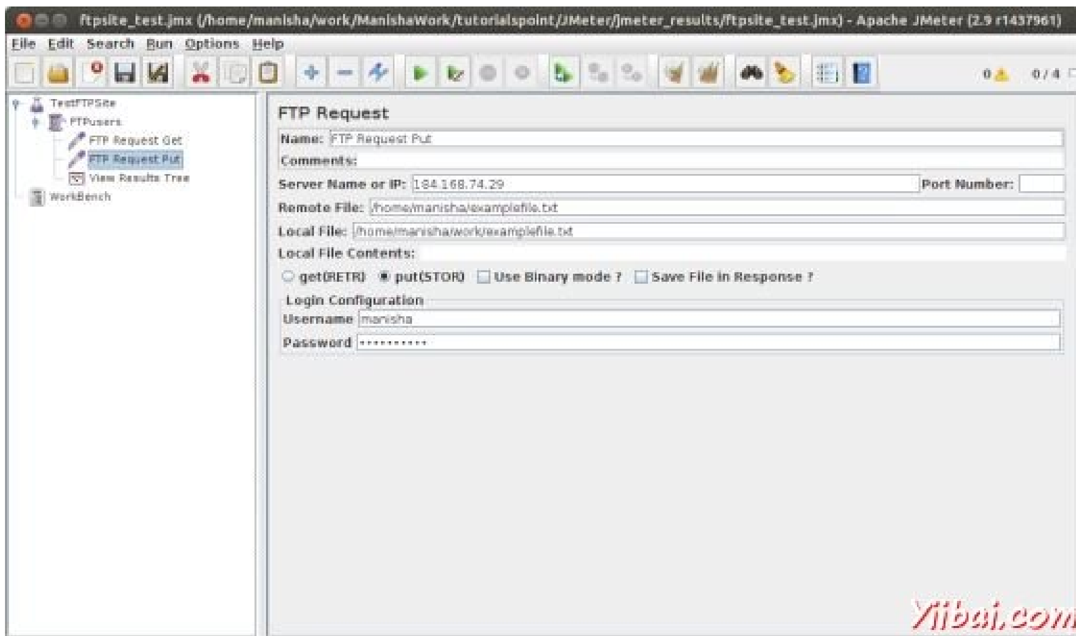
Local File:sample_ftp.txt

Select get(RETR)

Username:manisha

Password:manispass32

现在添加另一个FTP请求，并在下面的图片编辑的属性：




这个元素中输入下列详细信息：

Name: FTP Request Put

Server Name or IP: 184.168.74.29

Remote File: /home/manisha/examplefile.txt

Local File: /home/manisha/work/examplefile.txt

Select put(STOR)

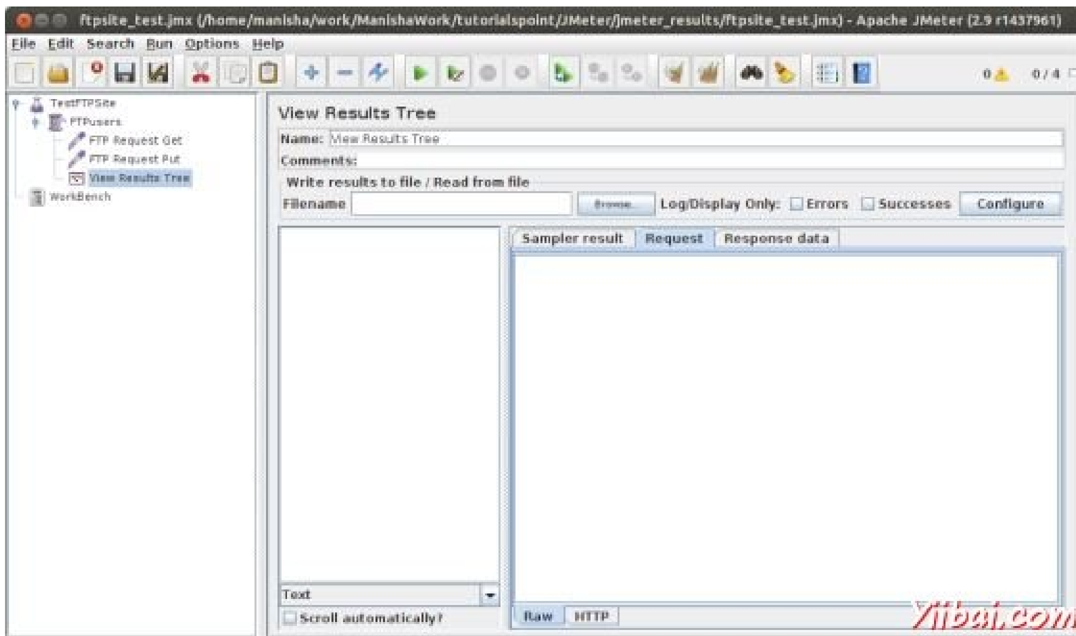
Username:manisha

Password:manisha123

## 添加监听器

需要添加到测试计划中的最后一个元素是一个监听器。此元素是负责为FTP请求的所有结果存储在一个文件中，呈现出可视化的数据模型。

选择 FTPusers 元素，并添加一个查看结果树监听器 (**Add > Listener > View Results Tree**).

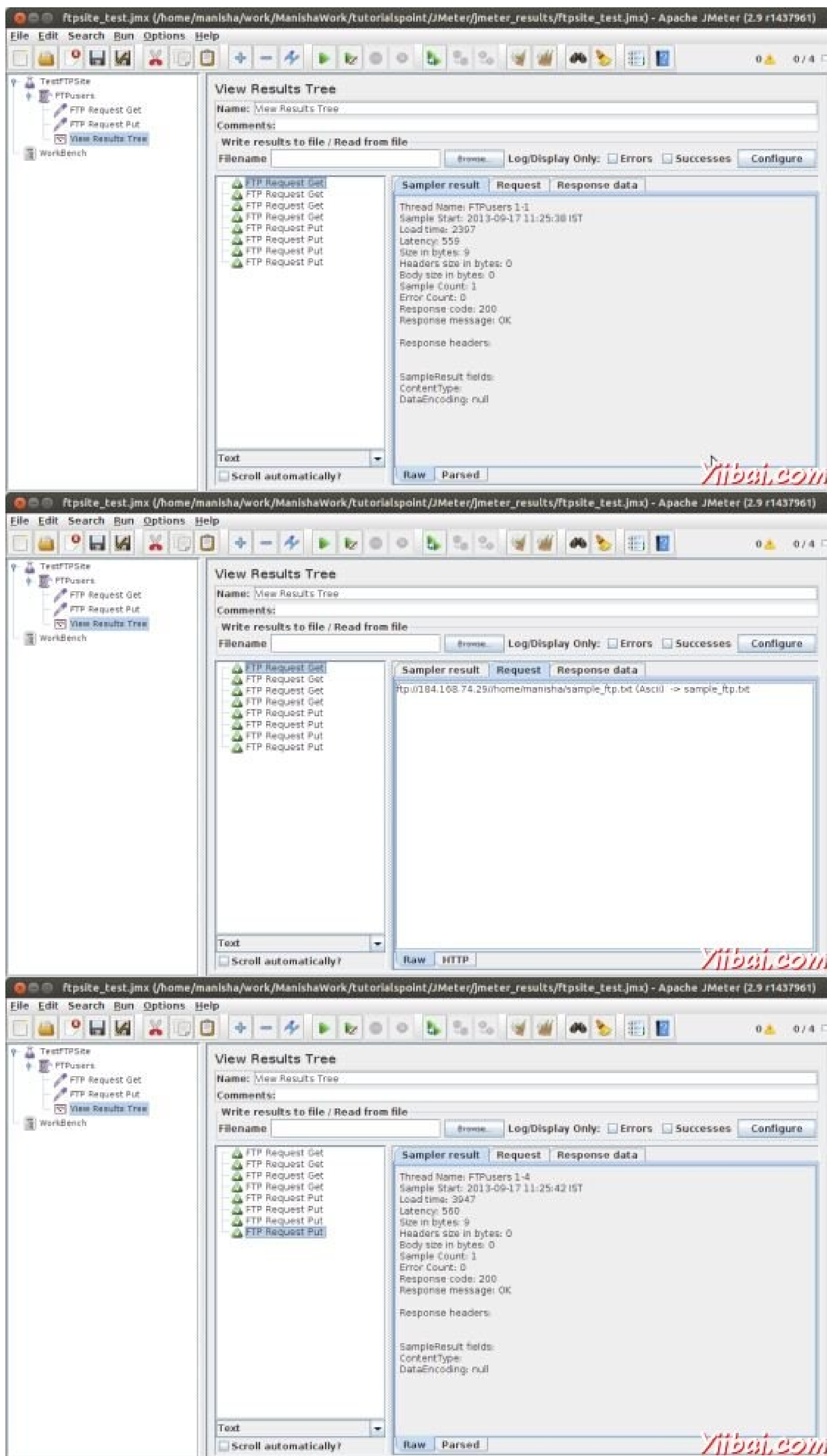



## 运行此测试计划

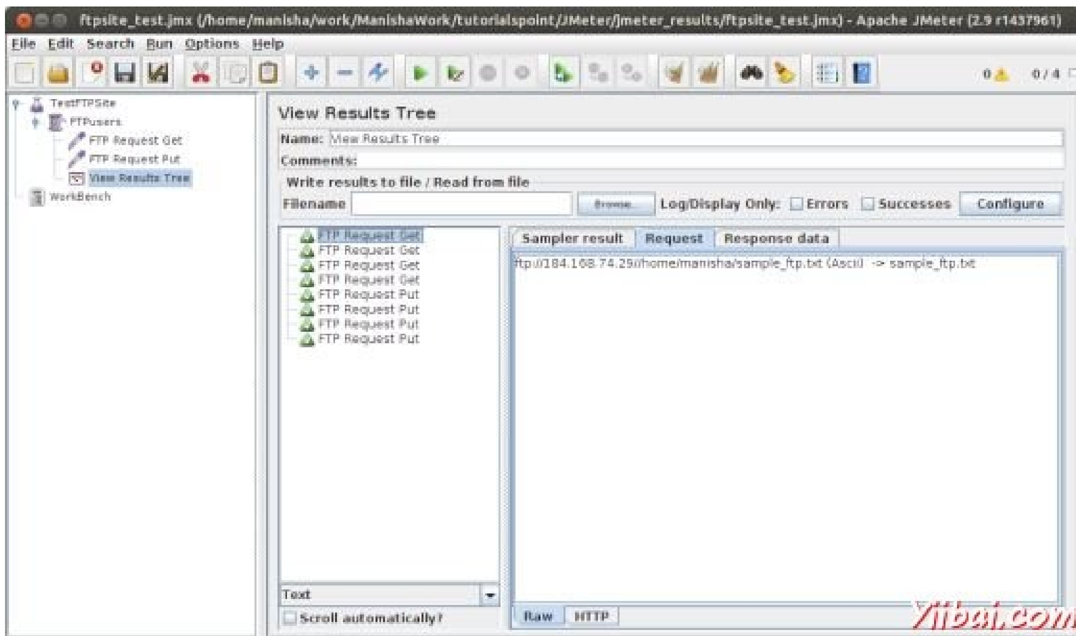
现在保存的以上测试计划 ftpsite_test.jmx。执行本测试计划使用 **Run > Start** 选项。

## 查看输出

下面的输出，在监听器中。



ftpsite_test.jmx (/home/manisha/work/ManishaWork/tutorialspoint/JMeter/jmeter_results/ftpsite_test.jmx) - Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
TestFTPSite
FTPusers
FTP Request Get
FTP Request Put
View Results Tree
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only:
Errors
Successes
Configure
Sampler result
Request
Response data
Thread Name: FTPusers 1-1
Sample Start: 2013-09-17 11:25:38 IST
Load time: 2307
Latency: 559
Size in bytes: 9
Headers size in bytes: 0
Body size in bytes: 0
Sample Count: 1
Error Count: 0
Response code: 200
Response message: OK
Response headers:
SampleResult fields:
ContentType:
DataEncoding: null
Text
Scroll automatically?
Raw
Parsed
ftp://184.168.74.29/home/manisha/sample_ftp.txt (Ascii) -> sample_ftp.txt
HTTP
Thread Name: FTPusers 1-4
Sample Start: 2013-09-17 11:25:42 IST
Load time: 3947
Latency: 560
yiibai.com



可以看到，四点要求的每个FTP请求测试成功。 GET请求检索文件存储在bin文件夹中。在我们的例子中，这将是 /home/manisha/apache-jmeter-2.9/bin/. PUT请求，上传文件的路径为 /home/manisha/.

# jMeter Webservice测试计划 - JMeter教程

在本章中，我们将学习如何创建一个测试计划，测试一个WebService。对于我们的测试目的，我们已经创建了一个简单的 Web 服务项目，并将其部署在 Tomcat 服务器上本地。

## 创建WebService项目

要创建一个 Web 服务项目中，我们使用了Eclipse IDE。首先编写服务端接口 HelloWorld 包下com.yiibai.ws。 HelloWorld.java 的内容如下：

```
package com.yiibai.ws;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{

@WebMethod String getHelloWorldMessage(String string);

}
```

这个服务有一个的方法 getHelloWorldMessage 需要字符串 参数。

下一步创建实现类 HelloWorldImpl.java 在包 com.yiibai.ws 下。

```
package com.yiibai.ws;

import javax.jws.WebService;

@WebService(endpointInterface="com.yiibai.ws.HelloWorld")
public class HelloWorldImpl  implements HelloWorld  {
    @Override
    public String getHelloWorldMessage(String myName){
        return("Hello "+myName+" to JAX WS world");
    }

}
```

作为下一个步骤，让本地发布此Web服务通过创建端点出版商和公开此服务的服务器上。

publish方法有两个参数：

- 端点URL字符串。
- 实现程序对象 HelloWorld 实现类，在这种情况下，这是作为一个Web服务公开在以上参数中提到的由URL标识的端点。

HelloWorldPublisher.java 内容如下所示：

```
package com.yiibai.endpoint;

import javax.xml.ws.Endpoint;

import com.yiibai.ws.HelloWorldImpl;

public class HelloWorldPublisher {

    public static void main(String[] args){
        Endpoint.publish("http://localhost:9000/ws/hello", new Hel
    }
}
```

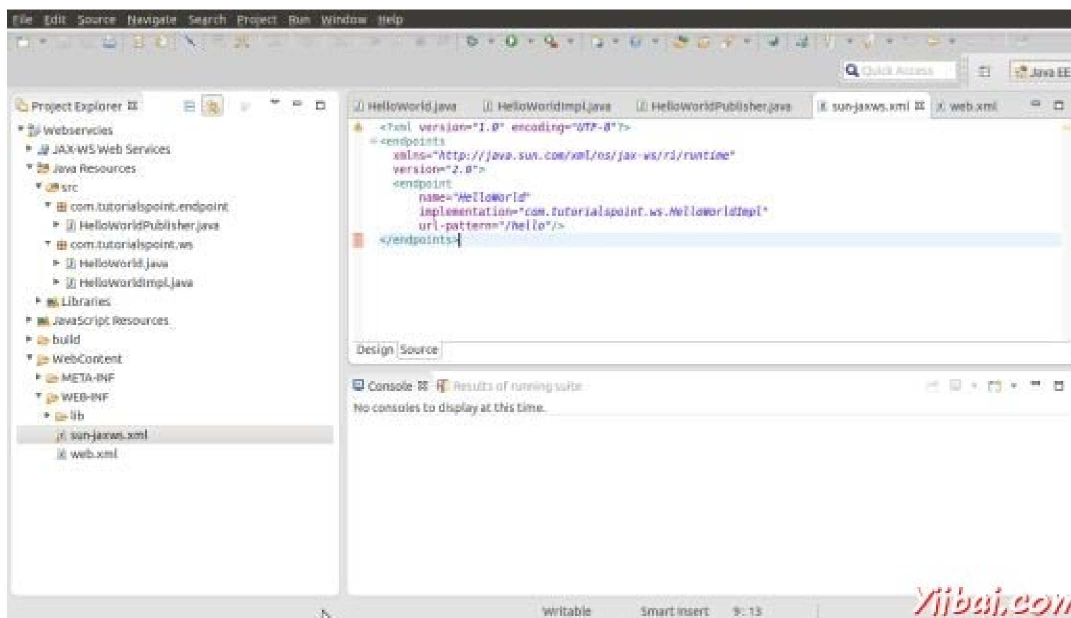
接下来修改web.xml如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
    <listener>
        <listener-class>
                com.sun.xml.ws.transport.http.servlet.WSServletCont
        </listener-class>
    </listener>
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>
            com.sun.xml.ws.transport.http.servlet.WSServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>120</session-timeout>
    </session-config>
</web-app>
```

要部署的 web 服务应用程序，我们需要 sun-jaxws.xml 配置文件，这个文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
  xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
  version="2.0">
  <endpoint
      name="HelloWorld"
      implementation="com.yiibai.ws.HelloWorldImpl"
      url-pattern="/hello"/>
</endpoints>
```

现在，所有文件都准备好目录结构看起来会像下面的图片：




现在，这个应用程序创建一个WAR文件。选择 **project > right click > Export > WAR**文件。hello.war 文件保存Tomcat 服务器 webapps文件夹下。现在启动Tomcat服务器。在服务器启动后，你应该能够访问web服务可以通过以下网址：

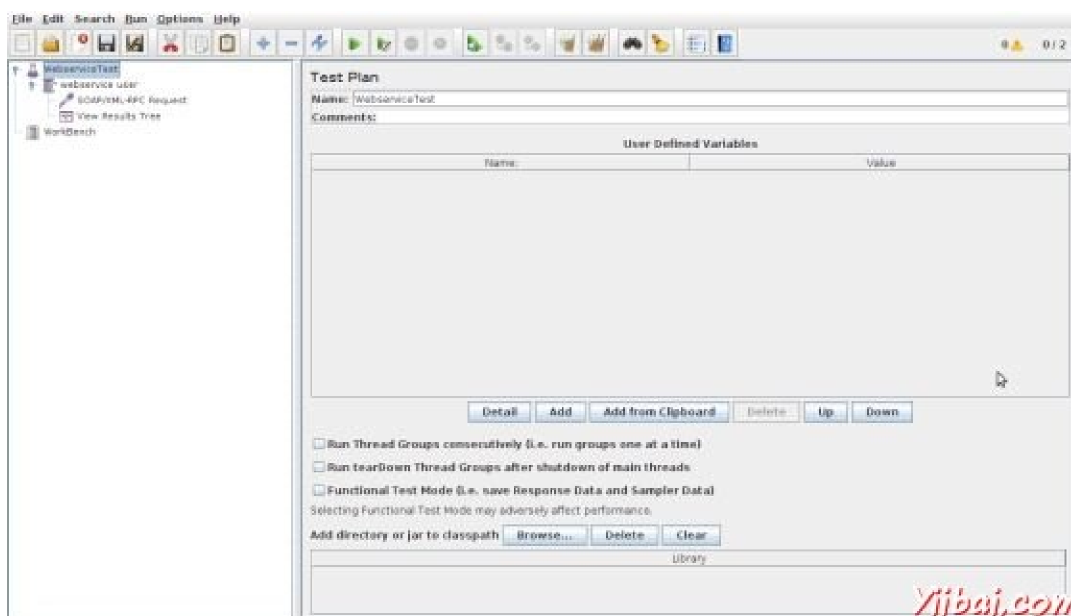
http://localhost:8080/hello/hello

# 创建JMeter测试计划

现在，让我们创建一个测试计划，测试上面的 web服务。

## 重命名测试计划

通过点击启动JMeter的窗口 /home/manisha/apache-jmeter-2.9/bin/jmeter.sh. 点击测试计划节点上。重命名此测试计划节点 WebserviceTest.



## 添加线程组

添加一个线程组，这是所有其他元素，如取样器，控制器，监听的占位符。右键单击我们的测试计划 **WebserviceTest(our Test Plan) > Add > Threads(Users) > Thread Group**。线程组将添加根据测试计划（**WebserviceTest**）的节点。

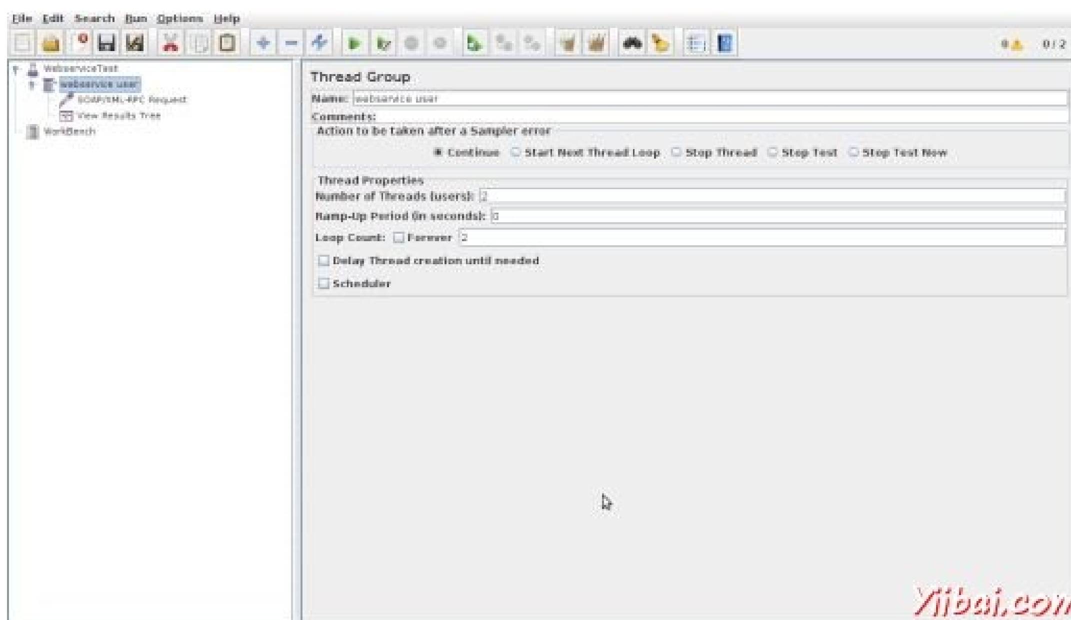
接下来，让我们修改线程组的默认属性，以满足我们的测试。改变以下属性：

Name: webservice user

Number of Threads (Users): 2

Ramp-Up Period: leave the the default value of 0 seconds.

Loop Count:2




## 添加SAMPLER-SOAP/XML RPC请求

现在，我们已经定义了用户，它是时间定义，他们将要执行的任务。我们将添加 SOAP/ XML-RPC 请求元素。点击鼠标右键得到添加菜单，然后选择 Add > Sampler > SOAP/XML-RPC Request，选择元素树中的SOAP/ XML-RPC请求，并在下面的图片编辑下列属性：

这个元素中输入下列详细信息：

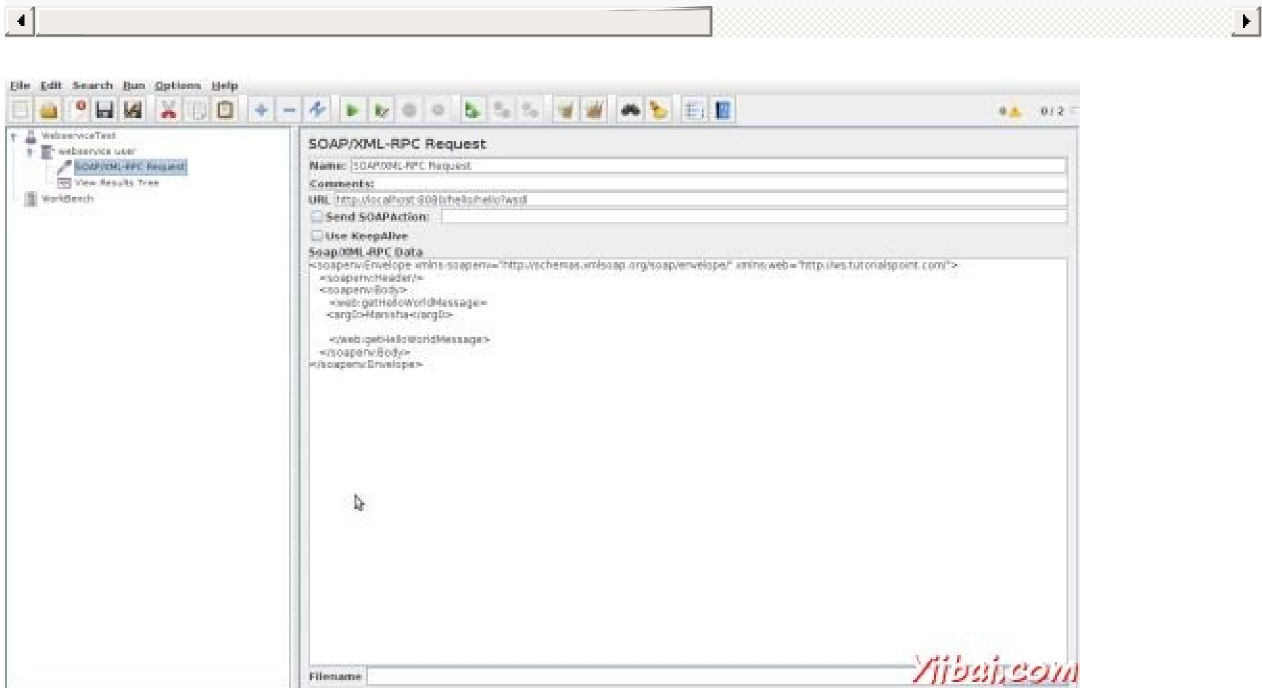
Name: SOAP/XML-RPC Request

URL: http://localhost:8080/hello/hello?wsdl

Soap/XML-RPC Data: Enter the below contents

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/en
   <soapenv:Header/>
   <soapenv:Body>
      <web:getHelloWorldMessage>
     <arg0>Manisha</arg0>

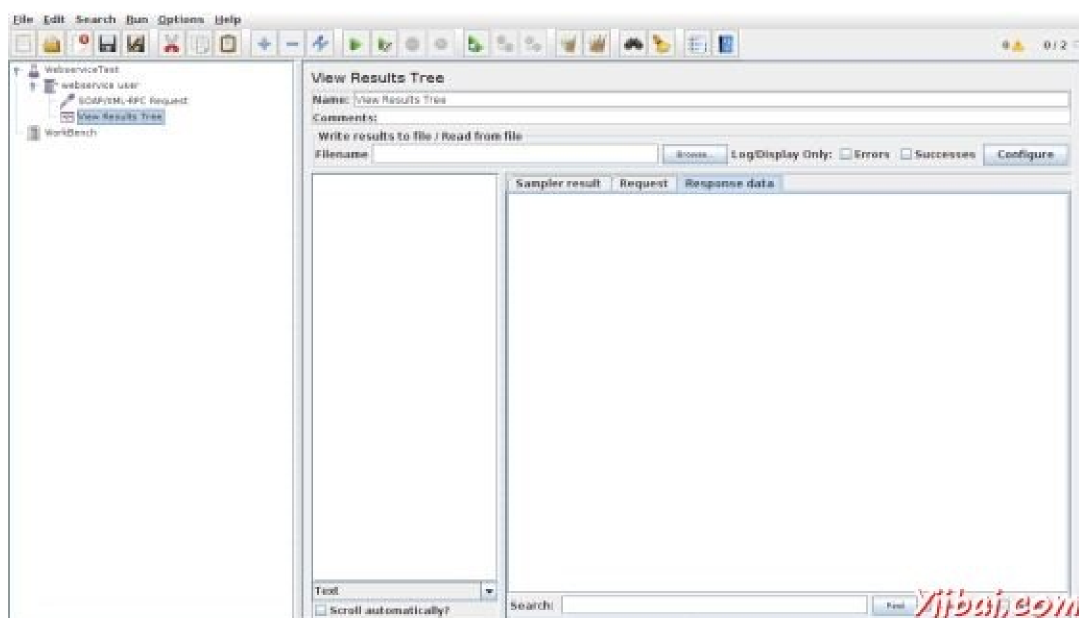
      </web:getHelloWorldMessage>
   </soapenv:Body>
</soapenv:Envelope>
```



## 添加监听器

需要添加到测试计划中的最后一个元素是一个监听器。此元素是负责所有的 HTTP 请求的结果存储在一个文件中，并呈现出可视化的数据模型。

选择 webservice 用户元素，并添加一个查看结果树监听器(**Add > Listener > View Results Tree**).

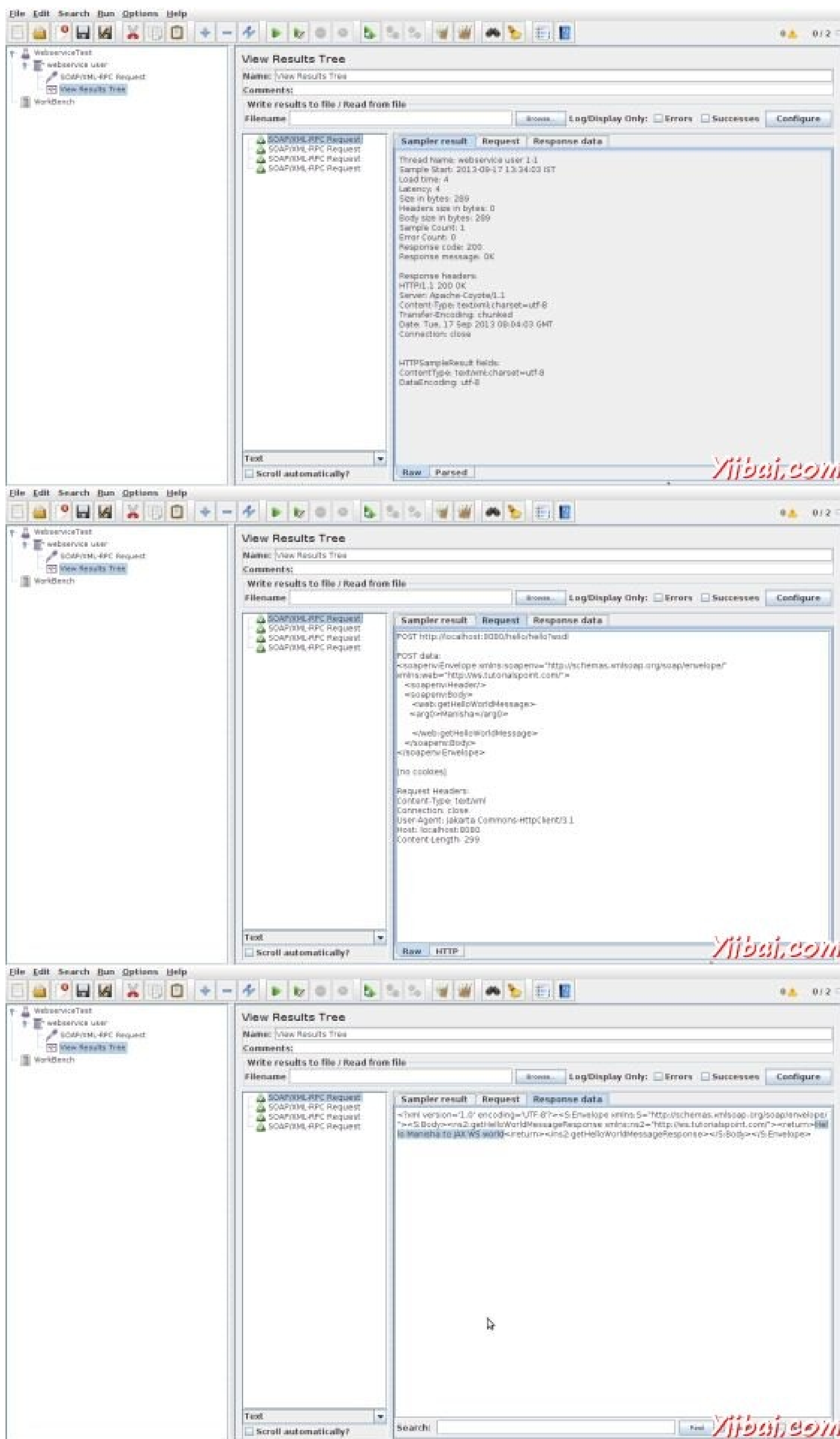



## 运行此测试计划

现在保存的以上测试计划 test_webservice.jmx。执行本测试计划使用 **Run > Start** 选项。

## 查看输出

下面的输出，可以看到在监听器中。




可以看到在过去的图像响应消息 "Hello Manisha to JAX WS world".

# jMeter JMS测试计划 - JMeter教程

在本章中，我们将学习如何写一个简单的测试计划，测试JMS（Java消息服务）。谁不知道JMS请阅读本章之前，让自己熟悉的JMS。 JMS支持两种类型的消息：

- 点到点消息: 一般用于交易发件人期望的响应队列消息。邮件系统是完全不同于普通的HTTP请求。在HTTP中，单个用户发送一个请求并得到响应。
- 主题消息：发布/订阅消息被俗称为主题的消息。主题消息一般使用情况下，消息发布由生产者和消费由多个用户。

让我们来看看这些为每个测试的例子。测试JMS的先决条件是：



- 我们将在我们的例子中使用的是Apache ActiveMQ。虽然也有其他JMS服务器，如IBM的WebSphere MQ（以前称为MQSeries），TIBCO等下载 Apache 网站的ActiveMQ二进制文件.
- 解压缩归档文件，请解压目录，运行以下命令从命令控制台启动 ActiveMQ 服务器：

  ```
  .inactivemq  start
  ```

  可以验证，如果 ActiveMQ 服务器已经开始在以下地址 http://localhost:8161/admin/ 访问管理界面。如果要求进行身份验证输入用户ID和密码为admin。屏幕将类似于如下：
- 现在复制activemq-all-x.x.x.jar（XXX的版本而定）从ActiveMQ的解压目录 /home/manisha/apache-jmeter-2.9/lib.

通过上述的设置，让我们构建的测试计划：

- JMS点到点测试计划
- JMS主题测试计划

# JMeter监视测试计划 - JMeter教程

在本章中，我们将讨论有关如何创建使用 JMeter 测试计划，监控Web服务器。利用监视器测试是：

- 监视器是有用的压力测试和系统管理。
- 用于压力测试，监视器服务器性能提供了额外的信息。
- 监视器可以更容易地看到在客户端服务器的性能和响应时间之间的关系.
- 系统管理工具，显示器提供了一个简单的方法来从一个控制台监视器多台服务器.

我们将需要一个Tomcat 5或以上版本进行监测。对于我们的测试目的，将监视Tomcat 7.0.42 服务器。可以测试任何servlet容器支持JMX（Java管理扩展）。让我们写一个测试案例监视器Tomcat 服务器。但在此之前，让我们先来设立 tomcat 服务器。

## 设置Tomcat服务器

我们先从“打开”Tomcat服务状态。要做到这一点，编辑的配置文件用户<TOMCAT_HOME>/conf/tomcat-users.xml. 此文件包含一个tomcat的用户部分（评论）如下：

```
<tomcat-users>

<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
</tomcat-users>
```

我们需要改变这部分，添加管理员的角色，manager，manager-gui和分配用户“admin”。修订后的文件如下：

```
<tomcat-users>

  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <role rolename="manager-jmx"/>
  <role rolename="manager-status"/>
  <user username="admin" password="admin" roles="manager-gui,manage

</tomcat-users>
```

现在启动 tomcat 服务器 <TOMCAT_HOME>/bin/startup.sh 在Linux平台 下，<TOMCAT_HOME>/bin/startup.bat 在Window. 一旦启动，检查Tomcat 监管工作进入下面的链接(在浏览器中打开)：

```
http://localhost:8080/manager/status?XML=true
```

验证窗口出现在浏览器中，进入tomcat 登录名和密码相关（在我们的案例中，它是admin）。然后，浏览器显示Tomcat作为下面的执行状态：

从上面的图片中，我们可以注意几件事情：

- 在URL中，XML = true（注意区分大小写）注意，可以清晰地显示JMeter 运作需要监视 Tomcat。
- 另外请注意，默认有两个连接器。加上Apache httpd 的mod_jk 前端模块，这是常用的 HTTP 连接器通过端口 8080 连接器直接访问到 Tomcat 的AJP连接器。

## 编写JMeter测试计划

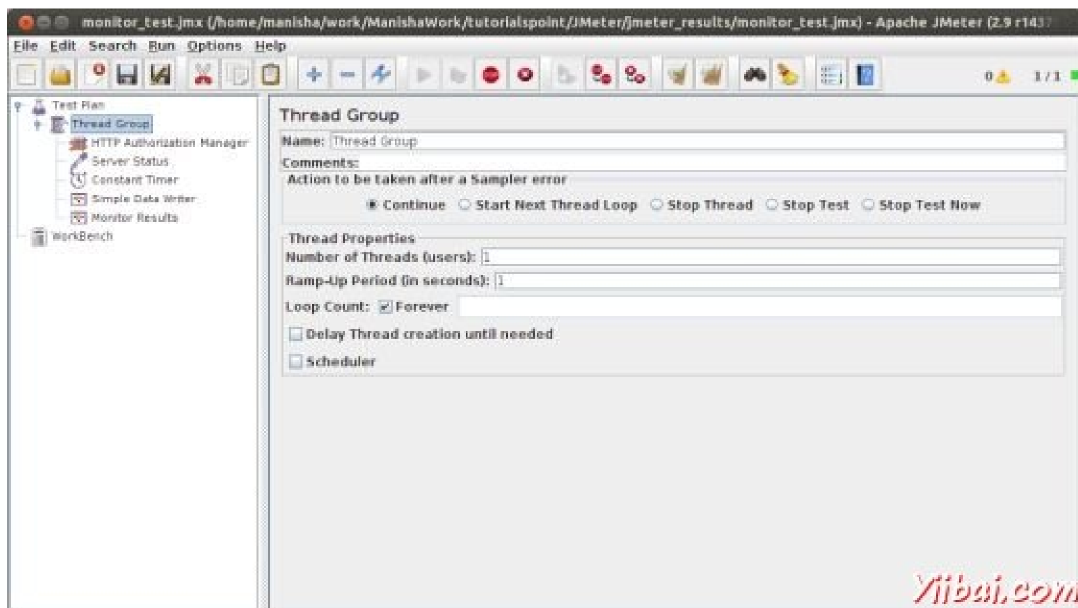
让我们监视器Tomcat服务器通过编写测试计划如下：

## 重命名测试计划

通过点击启动JMeter的窗口 /home/manisha/apache-jmeter-2.9/bin/jmeter.sh. 点击测试计划节点上。如解释在下一步添加一个线程组。

## 添加线程组

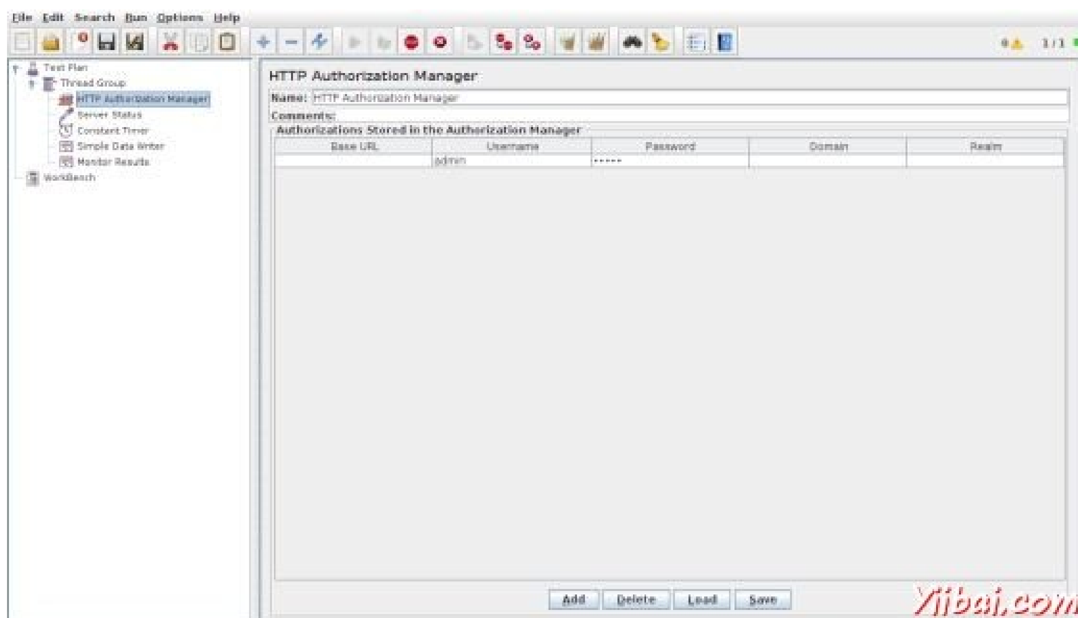
添加一个线程组。右键单击Test Plan > Add > Threads(Users) > Thread Group. 根据测试计划节点将添加线程组。改变永远循环计数（或大量），以便产生足够的样本。




## HTTP授权管理器

添加了HTTP授权管理到线程组元素 Add > Config element > HTTP Authorization Manager. 此元素管理认证要求浏览器中看到Tomcat服务器的状态。选择HTTP Authorization Manager 并编辑以下细节：

- Username : admin 管理员（取决于tomcat-users.xml文件中的配置）
- Password : admin（取决于配置tomcat-users.xml文件）
- 其他字段都留空。

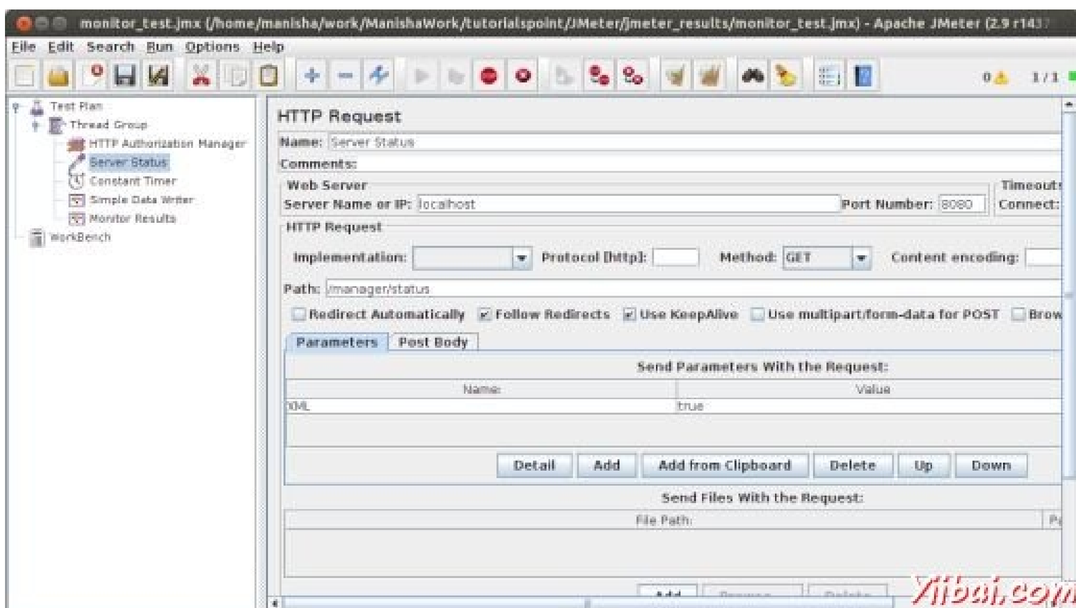



## 添加采样器的HTTP请求

现在，我们已经定义了我们的用户，它是时间来定义，他们将要执行的任务。我们将增加HTTP请求元素。点击鼠标右键得到添加菜单，然后选择Add > Sampler > HTTP Request. 然后，选择HTTP请求树中的元素，并在下面的图片编辑以下属性：

这个元素中输入下列详细信息：

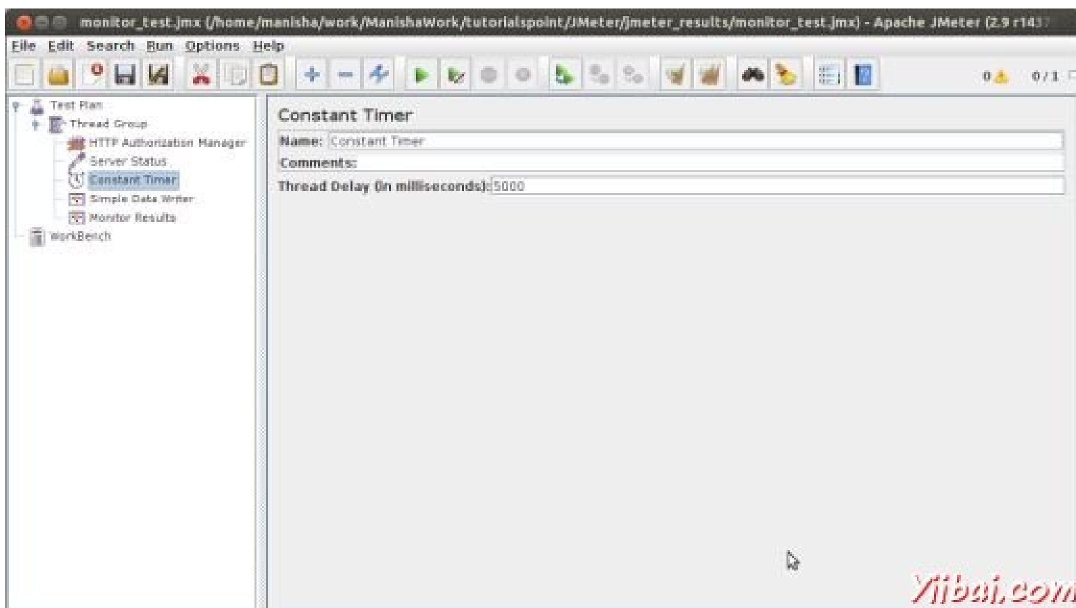
- Name : 服务器状态
- Server Name or IP : localhost
- Port : 8080
- Path : /manager/status
- Parameters : 添加请求参数名为“XML”大写。给它一个小写 “true” 值。
- Optional Tasks : 检查采样底部的“监视器”。




## 添加常量计时器

要定期要求服务器状态，添加的常量元素，这将使每个请求之间的时间间隔定时器。这个线程组中添加一个计时器 Add > Timer > Constant Timer.

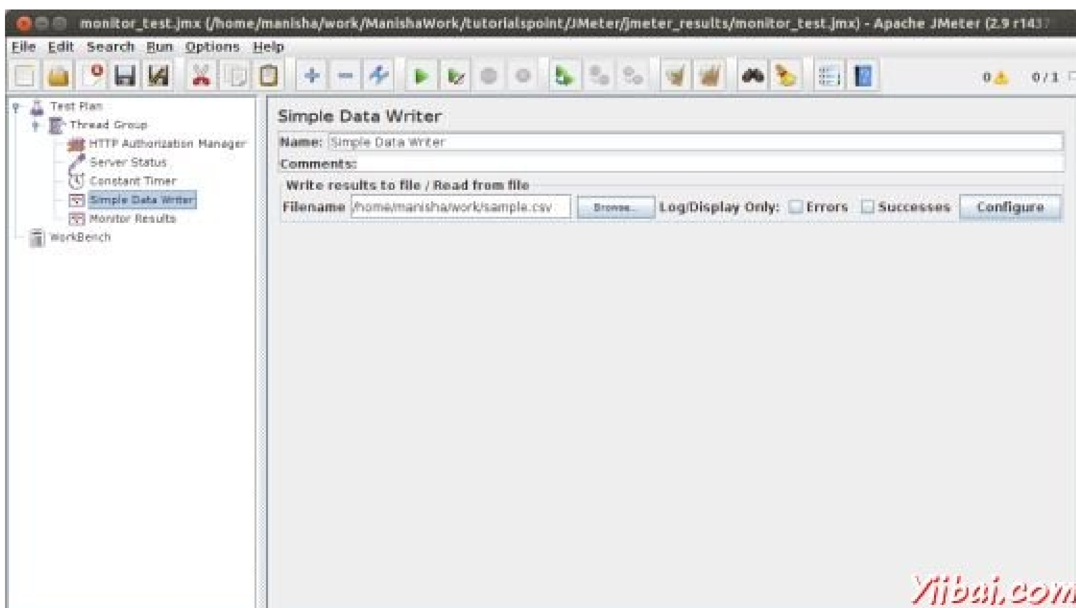
输入5000毫秒在线程的延迟框中。在一般情况下，使用短的间隔超过5秒将添加到服务器的压力。搞清楚什么是可接受的间隔，在生产环境中部署监视器。



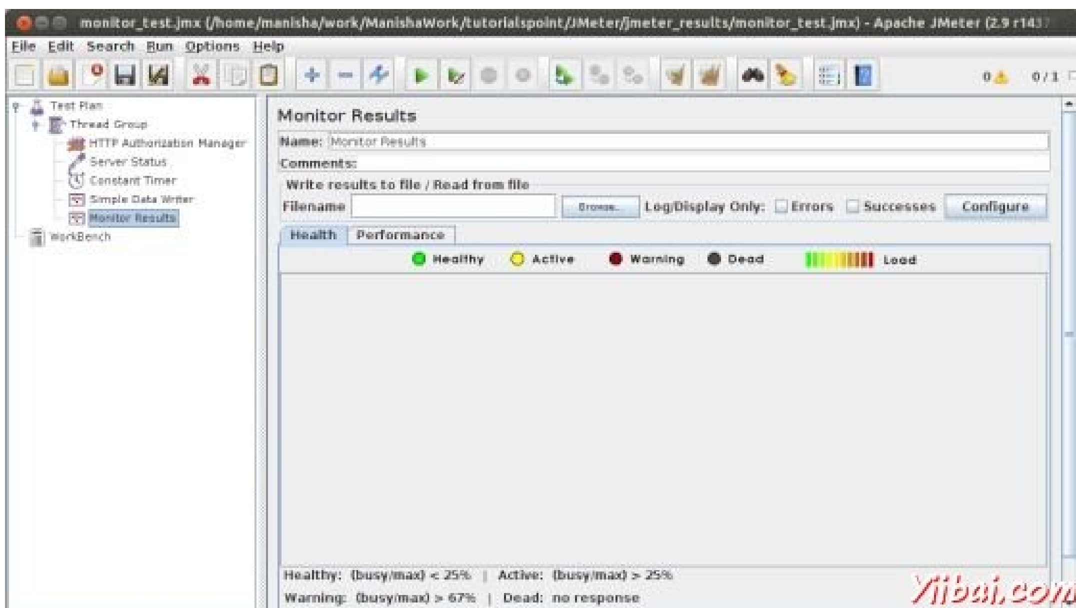

## 添加监听器

需要添加到测试计划中的最后一个元素是一个监听器。我们将两种类型的监听器。结果存储在一个文件中的第二个显示的图形视图中的结果。

选择线程组元素，并添加一个简单的数据写入器监听 Add > Listener > Simple Data Writer.下一步，指定输出文件的目录和文件名（在我们的案例中为 /home/manisha/work/sample.csv）




让我们添加另一个监听器，通过选择测试计划元件 Add > Listener > Monitor Results.

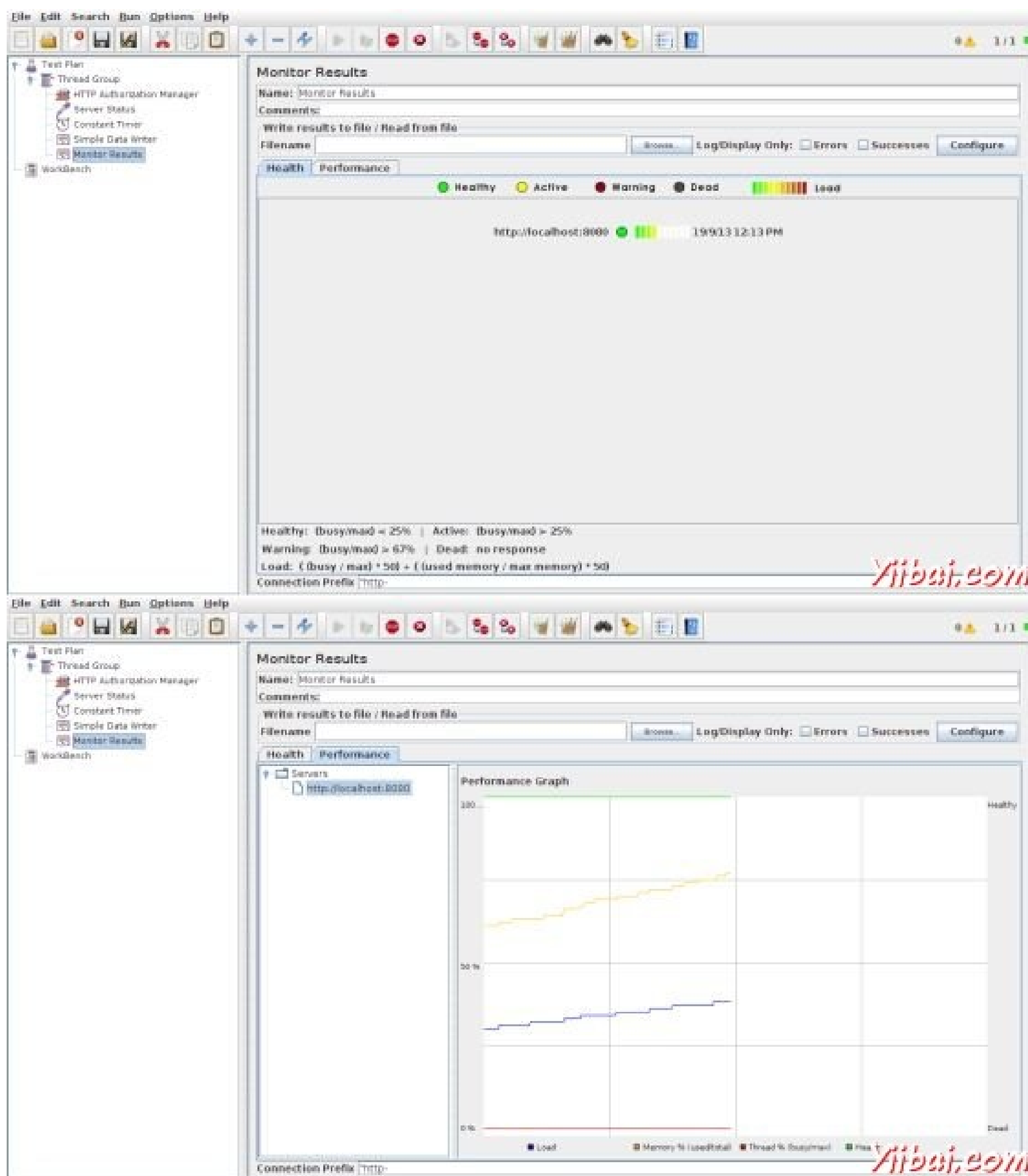



## 运行此测试计划

现在保存的以上测试计划 monitor_test.jmx 。执行本测试计划使用 Run > Start 选项.

## 查看输出

结果将被保存在 /home/manisha/work/sample.csv 的文件。在下面的图片，你还可以看到一个图形化的结果的监测结果监听




请注意图中有字幕图形的两侧上。在左边是%，右边是dead/healthy。如果记忆线尖峰迅速下降，这可能表明内存颠簸。在这些情况下，与Borland Optimizeit 的JProbe的分析应用程序。希望看到的是一个普通的负载，内存和线程模式。任何不稳定的行为通常表明表现欠佳或某种形式的错误。

# jMeter监听器 - JMeter教程

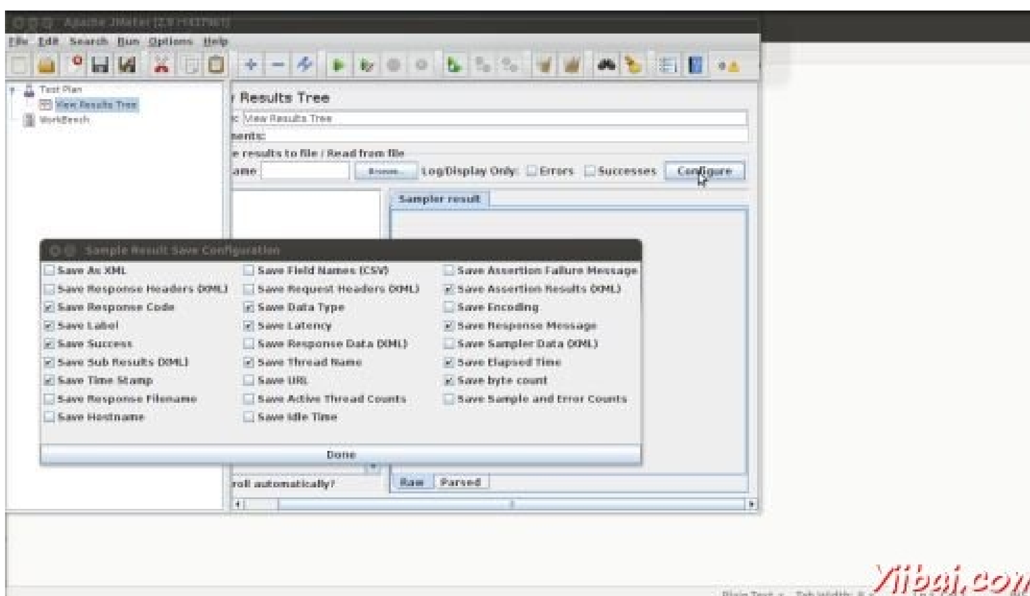
监听器提供 JMeter 有关JMeter的运行测试用例，同时收集信息的访问。结果还是监听器所收集的信息可以显示的形式：

- 树（tree）
- 表（tables）
- 图形（graphs）
- 日志文件

当它被指定，所有侦听器相同的原始数据写入到输出文件中。

## 默认配置

要保存的默认项可以定义：




- 在jmeter.properties（或user.properties）文件中。JMeter 的 /bin文件夹中，这个文件是存在。要更改默认格式，发现以下行 jmeter.properties 文件：

  ```
  jmeter.save.saveservice.output_format=
  ```

- 或通过使用配置弹出下面的图片所示：

JMeter 创建JTL（JMeter的文本日志）试运行的结果。这些通常被称为 JTL 文件，因为这是默认的扩展名 - 但可以使用任何扩展名。

如果使用相同的输出文件名，那么 JMeter 自动追加新的数据到文件末尾的运行多个测试。

监听器器可以记录到一个文件中，而不是到 UI 结果。它的目的是记录数据 GUI 的开销，从而提供一种有效的手段。

当运行在：

- GUI模式：使用监听器器简单的数据写入
- 非GUI模式：-l标志，可用于创建数据文件。

监听器可以使用大量的内存，如果有很多的样品。为了尽量减少所需的内存量，使用简单的数据写入，并使用CSV格式。

# CSV日志格式

CSV格式的日志格式取决于数据项中选择配置。只有指定的数据项被记录在文件中。外观上的列的顺序是固定的，如下所示：

| 字段 | 描述 | 示例值 |
|---|---|---|
| timeStamp | in milliseconds since 1/1/1970 | 1354223881017 |
| elapsed | in milliseconds | 1858 |
| label | sampler label | HTTP Request |
| responseCode | e.g. 200, 404 | 200 |
| responseMessage | e.g. OK | OK |
| threadName | Thread Group 1-1 | |
| dataType | e.g. text | text |
| success | true or false | true |
| failureMessage | if any | |
| bytes | number of bytes in the sample | 34908 |
| grpThreads | number of active threads in this thread group | 1 |
| allThreads | total number of active threads in all groups | 1 |
| URL | http://yiibai.com | |
| Filename | if Save Response to File was used | |
| latency | time to first response | 132 |
| encoding | utf-8 | |
| SampleCount | number of samples (1, unless multiple samples are aggregated) | 1 |
| ErrorCount | number of errors (0 or 1, unless multiple samples are aggregated) | 0 |
| Hostname | where the sample was generated | LaptopManisha |
| IdleTime | number of milliseconds of 'Idle' time (normally 0) | |
| Variables | if specified | |

# 保存响应数据

响应数据可以被保存在XML中的日志文件（如果需要）。但它也有限制，当文件很大，也不能包含图像。在这种情况下，使用后处理器Save_Responses_to_a_file。这会产生一个新的文件，对每个样品，样品保存的文件名。样品中的日志输出的文

件名可以被包含。这些数据将被从文件示例日志文件时，如果有必要被重新检索。

## 数据加载（读取）响应数据

要查看现有结果文件，可以使用“文件”浏览...“按钮选择一个文件。如果有必要，只需创建一个虚拟测试计划，在合适的监听器。

## GUI数据保存监听器

JMeter是能够保存任何监听器器作为一个PNG文件。要做到这一点，在左侧面板中选择监听器， Edit > Save As Image.

会出现一个文件对话框。输入所需的名称和保存的监听器。

# JMeter函数 - JMeter教程

## JMeter函数和用户变量

- JMeter 函数是可以填充任何采样器或其他元素在测试树字段的特殊值。一个函数调用看起来像这样：

```
${__functionName(var1,var2,var3)}
```

- _functionName 函数的名称相匹配。例如 ${__threadNum}.
- 如果一个函数参数包含一个逗号，然后转义“”，如下图所示。

```
${__time(EEE, d MMM yyyy)}
```

- 引用变量：

```
${VARIABLE}
```

### 函数列表

下表列出了一组函数类型大致分为：

| 函数类型 | 名称 | 注释 |
|---|---|---|
| Information | threadNum | get thread number |
| Information | samplerName | get the sampler name (label) |
| Information | machineIP | get the local machine IP address |
| Information | machineName | get the local machine name |
| Information | time | return current time in various formats |
| Information | log | log (or display) a message (and return the value) |
| Information | logn | log (or display) a message (empty return value) |
| Input | StringFromFile | read a line from a file |

| Input | CSVRead | read from CSV delimited file |
|---|---|---|
| Input | XPath | Use an XPath expression to read from a file |
| Calculation | counter | generate an incrementing number |
| Calculation | intSum | add int numbers |
| Calculation | longSum | add long numbers |
| Calculation | Random | generate a random number |
| Calculation | RandomString | generate a random string |
| Calculation | UUID | generate a random type 4 UUID |
| Scripting | BeanShell | run a BeanShell script |
| Scripting | javaScript | process JavaScript (Mozilla Rhino) |
| Scripting | jexl, jexl2 | evaluate a Commons Jexl expression |
| Properties | property | read a property |
| Properties | P | read a property (shorthand method) |
| Properties | setProperty | set a JMeter property |
| Variables | split | Split a string into variables |
| Variables | V | evaluate a variable name |
| Variables | eval | evaluate a variable expression |
| Variables | evalVar | evaluate an expression stored in a variable |
| String | regexFunction | parse previous response using a regular expression |
| String | escapeOroRegexpChars | quote meta chars used by ORO regular expression |
| String | char | generate Unicode char values from a list of numbers |
| String | unescape | Process strings containing Java escapes (e.g. & ) |
| String | unescapeHtml | Decode HTML-encoded strings |
| String | escapeHtml | Encode strings using HTML encoding |
| | | |

| String | TestPlanName | Return name of current test plan |
|---|---|---|

- 有两种类型的函数：
  - 用户定义的静态值（或变量）
  - 内置函数
- 用户定义的静态值允许用户定义变量时被替换为静态的值测试树编译并提交运行。
- 需要注意的是，变量目前无法嵌套，即${Var${N}}不起作用。
- **V**（变量）函数（版本**2.2**后）可用于执行此操作**: ${**V(Var${N})}.
- 这种类型的替换可能没有函数，但更方便，更直观

## 函数和变量可以在哪里使用？

- 函数和变量可以被写入到任何领域的任何测试部件。
- 下面的函数测试计划应确定工作：
  - intSum
  - longSum
  - machineName
  - BeanShell
  - javaScript
  - jexl
  - random
  - time
  - property functions
  - log functions

> 测试计划使用的功能有一些限制。JMeter的线程变量没有被完全成立处理功能时，不会设置变量名作为参数传递，将无法正常工作和变量引用，所以split() 和 regex() 变量赋值函数不会工作。 threadNum() 函数将不能工作（没有任何意义在测试计划级别）。

## 引用变量及函数

- 引用的变量中的测试元件是通过包围在变量名 '${' and '}'.

- 函数中引用同样的方式，但按照惯例，函数的名称以“__”开头，以避免冲突与用户值的名称。
- 有些函数带参数的配置，而这些括号中，逗号分隔。如果函数没有参数，括号可以省略。对于例如：

```
${__BeanShell(vars.put("name","value"))}
```

- 另外，可以定义你的脚本作为一个变量，例如测试计划：

```
SCRIPT     vars.put("name","value")
```

- 然后，该脚本可以被引用如下：

```
${__BeanShell(${SCRIPT})}
```

## 函数辅助对话框

JMeter 选项“tab“可从函数助手对话框。

- 使用函数助手，可以选择一个函数从拉下来，并指派其参数值。左边的表中的列的参数，简要说明和右列是你写在该参数的值。不同函数的不同参数。
- 一旦这样做了，点击“生成”按钮，并产生相应的字符串复制粘贴到测试计划。

## 预定义变量

JMeter 内部定义的一些变量。它们分别是：

- COOKIE_cookiename - 包含cookie的值
- JMeterThread.last_sample_ok - 与否的最后一个样本是确定的 - true/false。注：这是更新后的后处理和断言已经运行。
- START 变量

## 预先定义的特性

一些内置的的属性定义JMeter。下面列出了这些。为方便起见，启动的属性也被复制到具有相同名称的变量。

- START.MS - JMeter 启动时间（毫秒）
- START.YMD - JMeter 启动时间为 yyyyMMdd
- START.HMS - JMeter 启动时间为 HHmmss

- TESTSTART.MS - 测试开始时间（毫秒）

请注意，开始变量/属性代表 JMeter 的启动时间，而不是测试开始时间。它们主要适用于使用文件名等。

# jMeter正则表达式 - JMeter教程

使用正则表达式搜索和操纵文本，基于模式。 JMeter 解释正则表达式的形式或模式被用于整个JMeter 测试计划，包括模式匹配软件 Apache Jakarta ORO.

使用正则表达式，我们当然可以节省大量的时间，并实现更大的灵活性，因为我们建立或加强一个测试计划。正则表达式提供了一种简单的方法来获取信息页面时，它是不可能或很难预测结果。

使用表达式的标准用法的例子是从服务器响应得到一个会话ID。如果服务器返回一个唯一的会话密钥，我们可以很容易地得到我们的负载脚本中使用表达式。

要使用正则表达式在测试计划，需要使用正则表达式提取在JMeter。可以将正则表达式在测试计划中的任何组件使用。

这是值得强调的包含和比赛之间的差异，如用于响应断言测试元件：

- 包含表示正则表达式匹配至少有一些目标的一部分，所以'字母'"包含"'ph.b.因为正则表达式匹配'phabe"。
- 匹配正则表达式匹配整个目标。因此，"alphabet"是"匹配"'*t'。

假设想匹配一个Web页的以下部分：

```
name="file" value="readme.txt"
```

要提取readme.txt。一个合适的正则表达式如：

```
name="file" value="(.+?)">
```

上述的特殊字符是：

- ( and ) - 这些匹配字符串括起来的部分要返回
- . - 匹配任何字符
- 
    - 
        - 一次或更多次
- ? - 停止在第一个匹配成功时

## CREATE JMeter测试计划

让我们了解在正则表达式中使用正则表达式提取后处理器的元素，编写一个测试计划。此元素会从当前页面使用正则表达式识别文字图案所需的元素，符合提取文本。

首先，我们将写人名单和他们的电子邮件ID是一个HTML页面。它部署到 tomcat 服务器。 HTML（index.htmll）上的内容如下：

```
<html>
<head>
</head>
<body>
<table style="border: 1px solid #000000;">
<th style="border: 1px solid #000000;">ID</th><th style="border: 1p
<tr><td id="ID" style="border: 1px solid #000000;">3</td><td id="Na
<tr><td id="ID" style="border: 1px solid #000000;">4</td><td id="Na
</table>
</body>
</html>
```

部署在 Tomcat 服务器上，这个页面会看起来像在下面的快照：



在我们的测试计划中，我们将选择人以上列表页中看到的人表的第一行的人。为了捕捉这个人的ID，让我们首先确定的模式，我们会发现在第二排的人。在下面的快照中可以看出，第二个人的ID被包围<td id="ID">的和</TD>，它是具有这种模式的数据的第二行。我们可以用它来完全匹配的模式，我们希望从中提取信息。正如我们要提取两条信息从当前页的，该人ID和该人的姓名，字段定义如下：



启动JMeter，添加一个线程组 Test Plan > Add> Threads(Users)> Thread Group.

接下来，添加一个采样器的HTTP请求，右键单击选择的测试计划 Add > Sampler > HTTP Request 并进入详情如下：

- Name: Manage
- Server Name or IP: localhost
- Port Number: 8080
- Protocol: We will keep this blank, which means we want HTTP as the protocol.
- Path: jmeter/index.htmll



接下来，添加一个正则表达式提取。选择HTTP请求采样器（管理），右键点击 Add > Post Processor > Regular Expression Extractor.



上述快照详情如下：

| Field | 描述 |
| --- | --- |
| Reference Name | 所提取的测试将被存储在其中的变量的名称（refname）。 |
| Regular Expression | 对文本提取模式将匹配。文字组，将提取的字符'（'和'）'所包围。我们使用'+？'由<td..>..</ TD>标签包围的文本来表示一个单一实例。在我们的例子中的表达式为：<td id="ID">（+）</ TD> *S<td id="Name">（+）</ TD> S* |
| Template | 提取的文本的每个组将被放置作为成员变量person，“（”和“）”括起来的模式各组的顺序之后。每个组存储为refname_g#，其中refname是你输入的字符串作为参考名称，#是组号。$1$指组1，$2$是指第2组，$0$是指无论整个表达式匹配。在这个例子中，我们所提取的ID将被保持于Person_g1，而“名称”的值将被存储在Person_g2。 |
| Match No. | 既然我们打算只提取第二次出现的这种模式，相匹配的第二项，我们使用值2。值0将随机匹配，而在foreach控制器需要使用负值。 |
| Default | 如果该项目没有找到，这将是默认值。这是一个可选字段。可能会让它空白。 |

添加一个监听器来捕捉这个测试计划的结果。右键单击线程组选择 Add > Listener > View Results Tree 选项添加监听器。

保存测试计划为reg_express_test.jmx和运行测试。输出将是一个成功，因为在下面的快照：

# JMeter最佳实践 - JMeter教程

JMeter 尤其是当它运行在分布式环境中具有一定的局限性。遵循这些指导原则将有助于创造一个真正的和持续的负载：

- 使用 JMeter 多个实例的线程数较多的情况下。
- 检查的范围规则，并进行相应的设计。
- 总是使用命名约定的所有元素。
- 检查默认浏览器的连接设置，执行脚本之前。
- 添加适当监听器。
- 下面是一些建议，以减少资源的要求：
    - 使用非GUI模式: jmeter -n -t test.jmx -l test.jtl.
    - 使用为监听器尽可能少;如果使用-l标志如上，他们都可以被删除或禁用。
    - 禁用“查看结果树”监听器，因为它消耗了大量的内存，并可能导致在控制台冻结或JMeter的运行内存。它是，但是，安全使用“查看结果树”监听器只用“错误”检查。
    - 而不是使用很多类似的采样，在一个循环中使用相同的采样和使用变量的（CSV数据集），以不同的样品。或许使用访问日志取样。
    - 不要使用功能模式。
    - 使用CSV输出，而不是XML。
    - 只保存你需要的数据。
    - 使用尽可能尽可能少的断言。
    - 禁用所有的JMeter图，因为他们消耗了大量的内存。可以查看所有在Web界面使用JTLs标签的实时图形。
    - 不要忘了删除的本地路径设置配置如果使用CSV数据。
    - 每次测试运行前清理“文件”选项卡。

# JOGL教程

本章介绍了OpenGL，Java OpenGL绑定（GL4java，LWJGL，JOGL）和JOGL比其他的OpenGL的优点。

Java支持OpenGL（JOGL）是近期在Java OpenGL图形API结合。它是一个包装库，它可以访问OpenGL API，并且它被设计来创建Java编码的2D和3D图形应用程序。 JOGL是前麻省理工学院的研究生肯·拉塞尔和克里斯·克莱恩最初开发的一个开源库。后来发布到Sun Microsystems，现在它是Java图形和音频处理（JOGAMP）。用于各种操作系统，如Windows，Solaris和Mac OS X和Linux（基于x86）JOGL功能。

## OpenGL是什么?

OpenGL代表开放图形库，用来创建2D和3D图形的集合。在OpenGL中，可以创建一个使用非常基本的图元，如点，线，多边形，位图和图像复杂的三维形状。

下面是OpenGL的几个特点：

- 它可以在多个平台上工作。
- 它有几种语言，如C ++，Python绑定等。
- 它可以呈现2D和3D矢量图形。
- 它与图形处理单元（GPU）实现快速，高品质的渲染。 （渲染是指创建从一个二维或三维模型的图像的过程。）
- 它是用于编写3D图形应用程序的行业标准API。例如，游戏，屏幕保护程序等。
- 它包含约150个命令，程序员可以使用指定的对象和操作来开发应用程序。
- 它包含了OpenGL实用库（GLU），提供各种建模功能，如二次曲面和NURBS曲线。 GLU是OpenGL的一个标准组件。
- OpenGL的设计重点是效率，效益和实现使用多语言在多个平台。保持一个OpenGL API的简单框架，不包括窗口的任务。因此，OpenGL依赖于其他编程语言对加窗的任务。

## Java绑定OpenGL API

它是一个Java规范请求（JSR）的API规范，它允许使用OpenGL在Java平台上。

| 产品规格 | 详细 |
|---|---|
| JSR 231 | Java绑定包支持Java SE平台。 |
| JSR 239 | Java绑定包支持Java ME平台。 |

在Java中有各种OpenGL的绑定。他们是：

### GL4java

这是被称为OpenGL的Java技术。它链接OpenGL1.3和几乎所有的供应商扩展。此外，它可以用于抽象窗口工具包（AWT）和摆动。它是一个游戏聚焦OpenGL结合，这是一个显示全屏幕应用程序的单个窗口。

### LWJGL

- 轻量级的Java游戏库（LWJGL），使用OpenGL1.5，并结合Java最新版本。
- 它可以使用JSE1.4的全屏功能。但它对于AWT/Swings的支持有限。
- 它适用于重量轻的设备，如移动电话，嵌入式设备等。

### JOGL

- JOGL只专注于2D和3D渲染。处理声音和输入输出的接口不包括在JOGL。
- 它包括图形工具库（GLU），GL实用工具包（GLUT），和自身API-native窗口工具包（NEWT）。

## 为什么要用JOGL?

- 它提供了完全访问的OpenGL API（版本1.0，4.3，ES1，ES2 ES3），以及几乎所有的供应商扩展。因此，在OpenGL中的所有功能都包含在JOGL。
- JOGL集成了AWT，Swing和标准窗口小部件工具箱（SWT）。它也包括它自己的本机窗口工具包（NEWT）。因此，它提供窗口的完整支持。

## JOGL历史

- 1992 - Silicon Graphics公司发布了第一个OpenGL的规范。
- 2003 - Java.net网站推出的新功能和JOGL发表首次在同一网站上。
- 2010 – 自2010年以来，它一直在BSD许可证下独立的开源项目，它是计算机软件一个自由的许可证。

# JOGL安装 - JOGL教程

本章介绍了设置环境以使用JOGL使用不同的集成开发环境(IDE)，在您的系统上。

# 安装JOGL

对于JOGL安装，需要有以下系统要求：

## 系统要求

第一个要求是要在机器上安装Java Development Kit（JDK）。

| 要求 | 描述 |
|---|---|
| JDK 版本 | 1.4 或以上 |
| 内存 | 没有最小限制 |
| 硬盘大小 | 没有最小限制 |
| 操作系统 | 没有最小限制 |

需要按照设置给定的步骤，从配置环境入手JOGL应用程序开发：

## 第1步 - 在机器上验证Java安装

系统的开放式控制台，并执行下面的Java命令：

| 平台 | 任务 | 命令 |
|---|---|---|
| Windows | 打开命令控制台 | C:\>java-version |
| Linux | 打开命令终端 | $java- version |
| MAC | 打开终端 | Machine:~ joseph$ java -version |

验证在各操作系统的输出。

| 平台 | 输出 |
|---|---|
| Windows | Java “1.6.0.21”java(TM) SE Runtime Environment(build 1..6.0_21-b07)Java HotSpot(TM) Client VM(buld 17.0-b7,mixed mode, sharing) |
| Linux | Java “1.6.0.21”java(TM) SE Runtime Environment(build 1..6.0_21-b07)Java HotSpot(TM) Client VM(buld 17.0-b7,mixed mode ,sharing) |
| MAC | Java “1.6.0.21”java(TM) SE Runtime Environment(build 1..6.0_21-b07)Java HotSpot(TM) Client VM(buld 17.0-b7,mixed mode ,sharing) |

## 第2步 - 安装Java开发工具包（JDK）

如果Java未在你的机器上安装，则需要从Oracle网站上下载安装Java SDK：www.oracle.com/technetwork/java/javase/downloads/. 您可以从下载的文件安装JDK说明。需要按照安装和配置设置的指示。最后，设置PATH和JAVA_HOME环境变量指向包含的java.exe和javac.exe文件的目录，它们分别为：java_install_dir/bin和java_install_dir。

设置Java-home环境变量指向的基本目录的位置相同的路径，安装在机器上的Java程序。

| 平台 | 命令 |
|---|---|
| Windows | 设置环境变量 JAVA_HOME to C:\ProgramFiles\Java\Jdk1.6.0_21 |
| Linux | Export JAVA_HOME=/usr/local/java-current |
| MAC | Export JAVA_HOME=/Library/Java/Home |

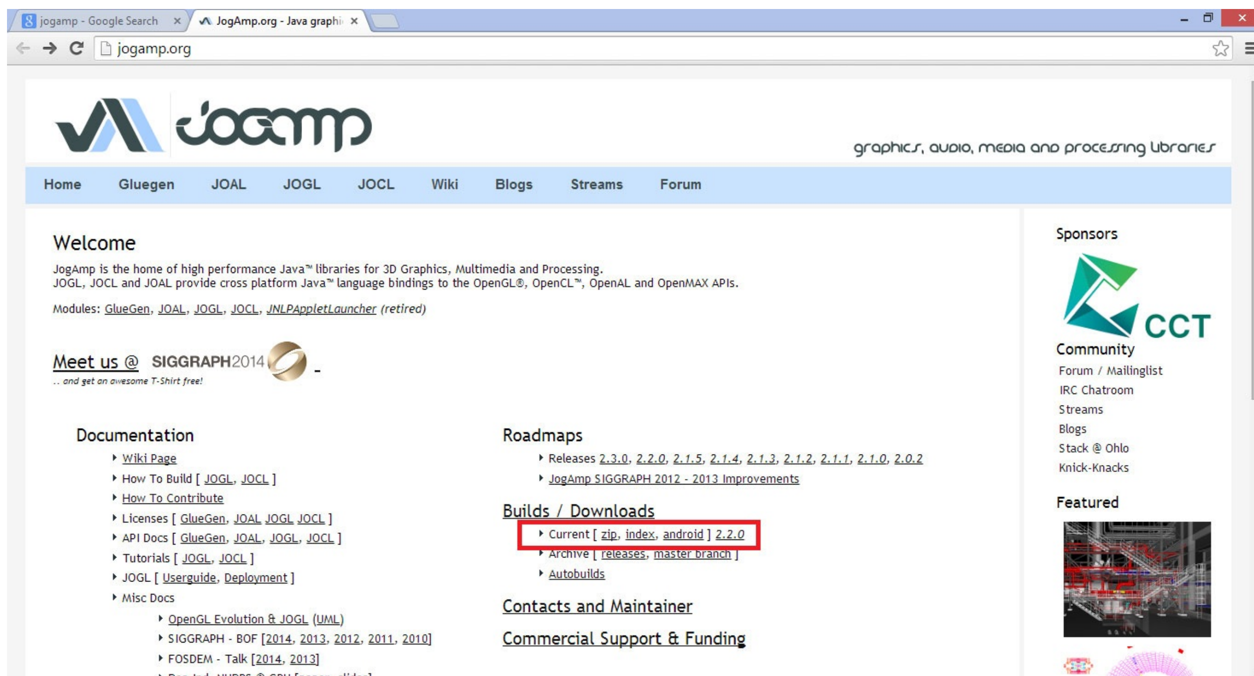
附加的Java编译器位置，系统路径如下：

| 平台 | 命令 |
|---|---|
| Windows | 添加字符串 ;%JAVA_HOME% bin at the end of the system variable and path |
| Linux | Export PATH=$PATH:$JAVA_HOME/bin/ |
| MAC | Not required |

## 第3步 - 下载JOGL

- 可以从网站上下载JOGL的最新版本 www.jogamp.org
- 前往主页 www.jogamp.org

- 点击 Builds/Downloads > Current (zip).




可以看到网站所维护的所有.jar文件的API列表。

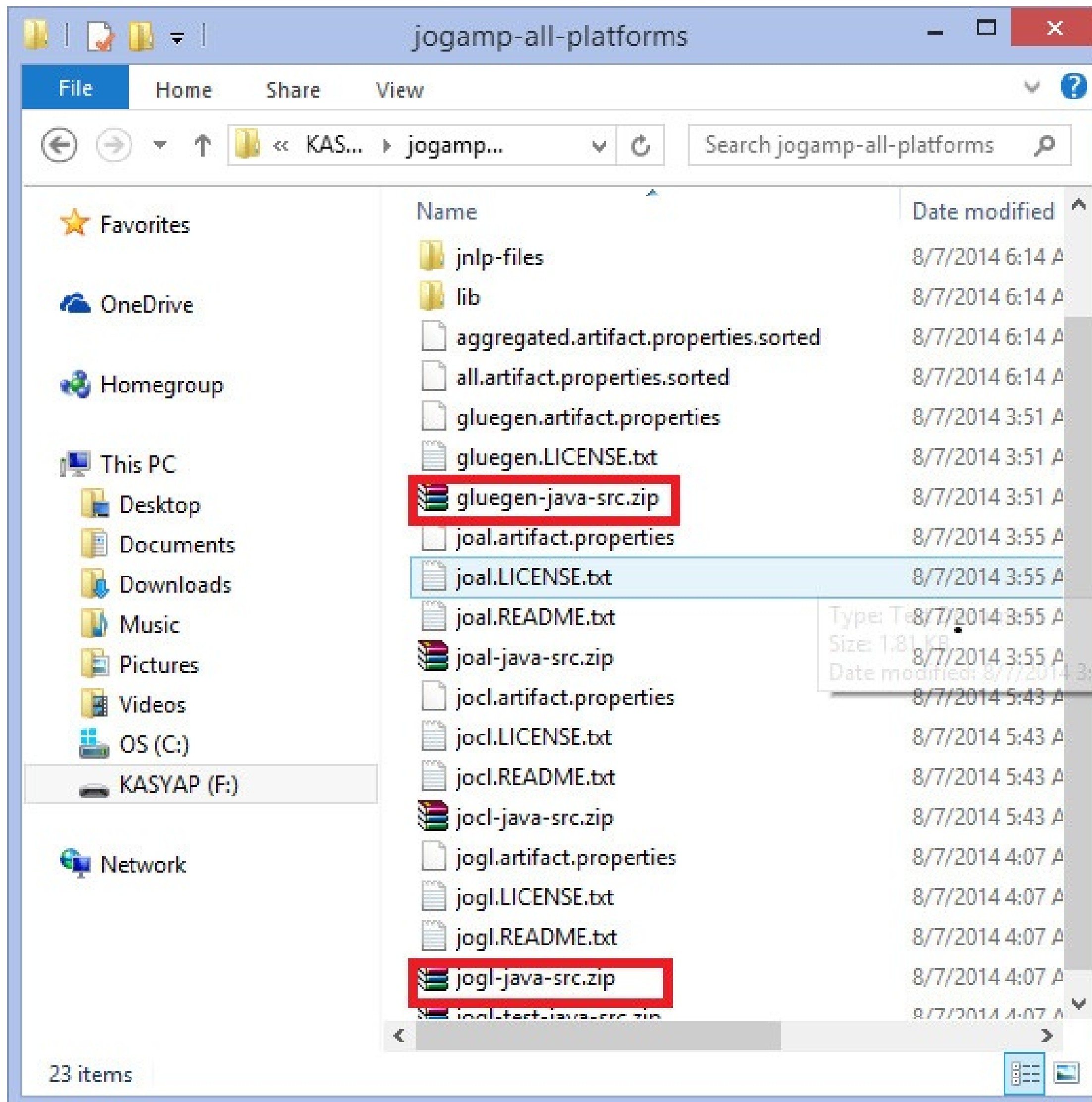
jogamp.org/deployment/jogamp-current/archive/

# Index of /deployment/jogamp-current/archive

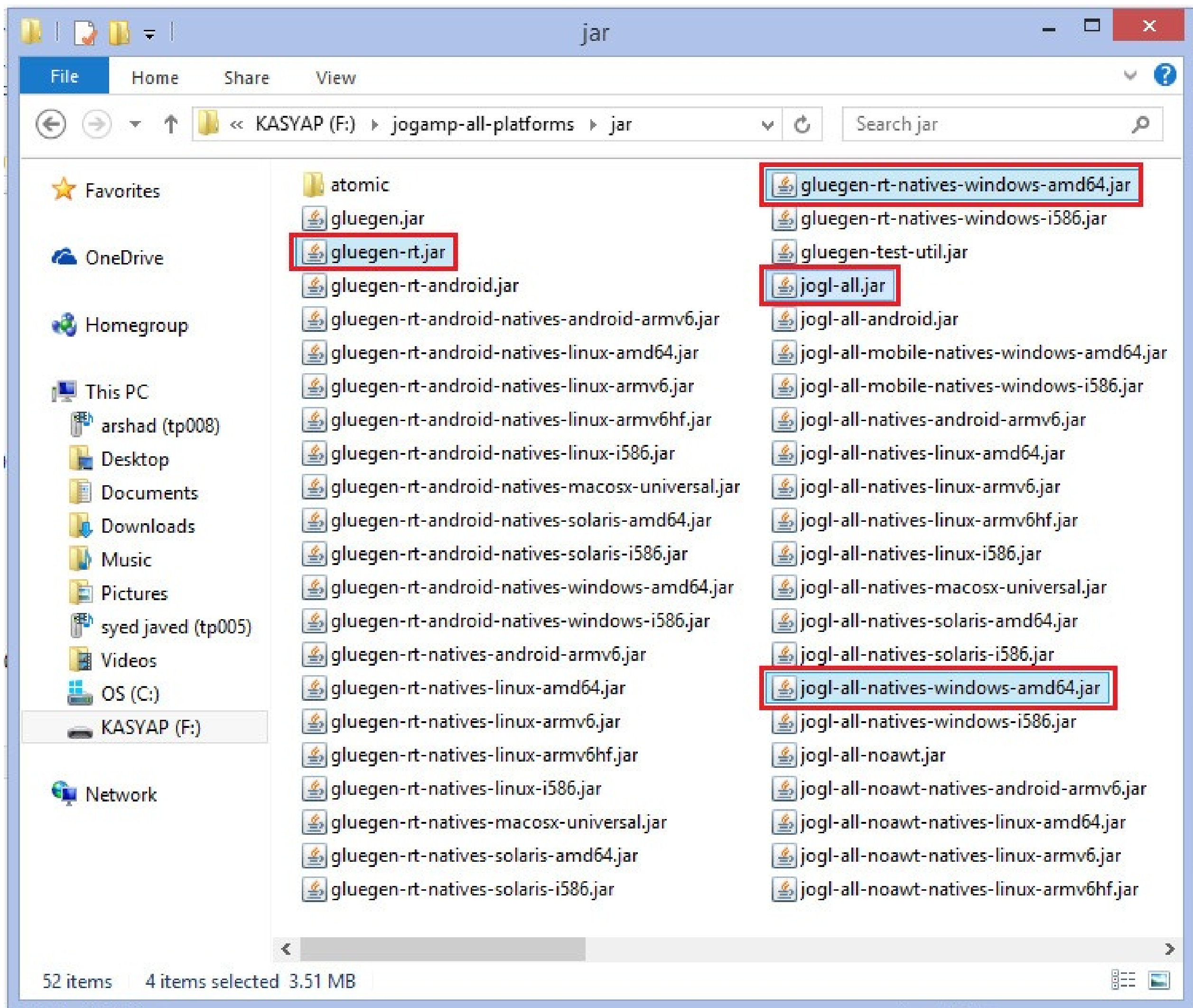
| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | | - | |
| ChangeLogs/ | 07-Aug-2014 03:12 | - | |
| Sources/ | 07-Aug-2014 03:13 | - | |
| gluegen-javadoc.7z | 07-Aug-2014 00:24 | 305K | |
| joal-demos.7z | 07-Aug-2014 02:43 | 1.2M | |
| joal-javadoc.7z | 07-Aug-2014 00:25 | 104K | |
| jocl-demos.7z | 07-Aug-2014 02:44 | 552K | |
| jocl-javadoc.7z | 07-Aug-2014 00:39 | 181K | |
| jogamp-all-platforms.7z | 07-Aug-2014 02:44 | 29M | |
| jogl-demos.7z | 07-Aug-2014 02:43 | 25M | |
| jogl-javadoc.7z | 07-Aug-2014 00:38 | 1.6M | |
| test-results/ | 07-Aug-2014 02:44 | - | |

*Apache/2.2.22 (Debian) Server at jogamp.org Port 80*

- 下载库.jar文件jogamp-all-platforms.7z，JAVA文档OpenGL库glugen-javadoc.7z和JOGL JOGL-javadocs.7z。
- 提取使用任何压缩解压软件下载的.jar文件。
- 当打开解压文件夹，会发现jar 文件夹，源代码和其他文件。




- 获取源代码gluegen-java-src.zip和jogl-java-src.zip支持IDE。这是可选的。
- 文件夹中的jar，有多个.jar文件。文件的集合属于Glugen和JOGL。
- JOAMP提供支持多种操作系统，如Windows，Solraris，Linux和Android原生库。因此，需要采取适当的jar文件，这些文件可以在需要的平台上执行。例如，如果使用的是Windows64位操作系统，那么可以通过jarfolder以下.jar文件：
- gluegenrt.jar
- jogl-all.jar
- gluegen-rt-natives-windows-amd64.jar
- jogl-all-natives-windowsamd64.jar



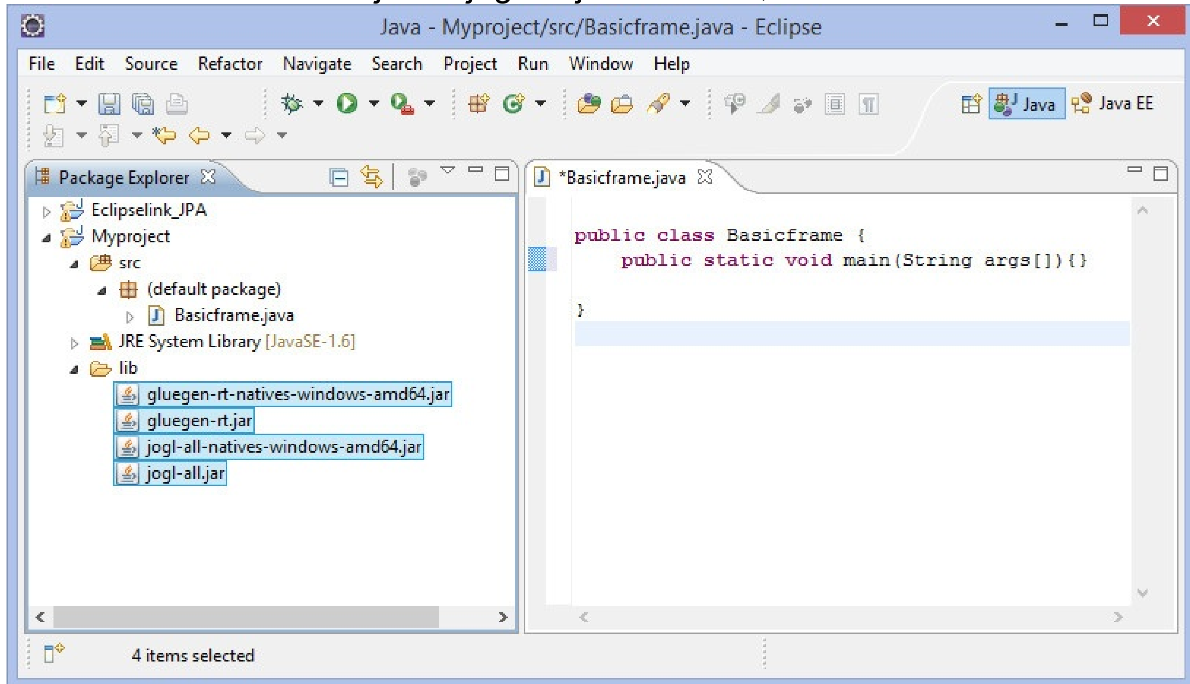

# Eclipse4.4设置JOGL

按照给定的程序设置JOGL：

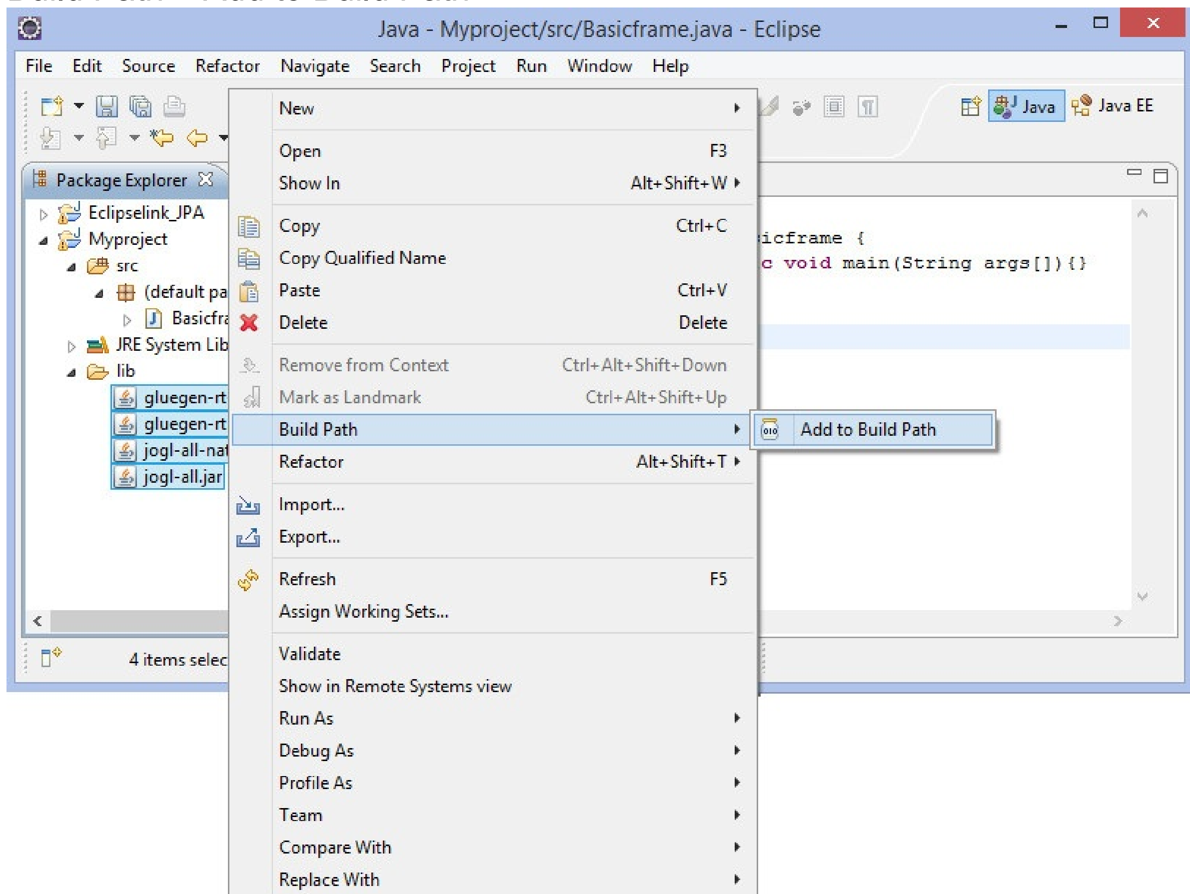
## 添加以下类库

1. 打开eclipse.
2. 创建一个新工程
3. 创建一个名为lib目录在项目文件夹中的新文件夹。

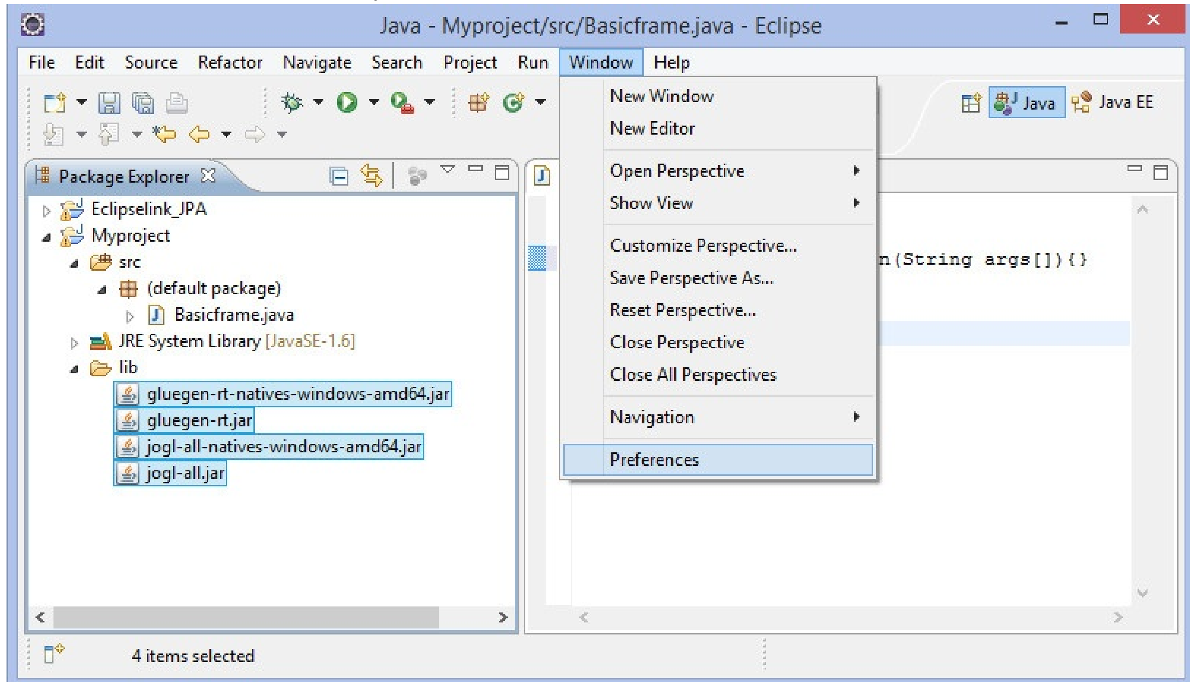
4. 复制文件 gluegen-rt-natives-windows-amd64.jar, gluegen-rt.jar, jogl-all-natives-windowsamd64.jar 和 jogl-all.jar 到 lib 目录。




5. 现在选择这些文件，然后右键单击鼠标按钮。将显示一个快捷菜单，其中包含 Build Path > Add to Build Path.



6. 为了可以在其他项目中可以使用所有的.jar文件，进入主菜单。选择 Window > Preferences. 出现在首选项窗口。




7. 在首选项窗口，在下拉菜单上的左侧菜单中，按照 hierarchy- Java-> Build Path -> User 库.
8. 点击 “New…” 按钮.
9. 这将打开一个对话框。输入库名称 jogl2.1.
10. 添加 jar 文件 glugen-rt.jar 和 jogl-all.jar 使用按钮 “Add External JARs...”.
11. 这将创建一个名为新的用户库 jogl2.1.

以同样的方式，我们可以为添加.jar文件添加的java文件和源代码。

## 添加本地库

1. 展开jogl-all.jar 的节点上，选择Javadoc位置（无）。
2. 点击 “New…” 钮，输入JOGL Java文件的名称。
3. 请点击 “Add External JARs…” 按钮。
4. 这将打开你需要选择JOGL Java文档，我们已经先前下载的位置的对话框。

## 添加源代码

1. 选择节点本地库的位置：(None).
2. 点击 “New…” 按钮.
3. 本地库和点击输入姓名 “OK” 按钮.

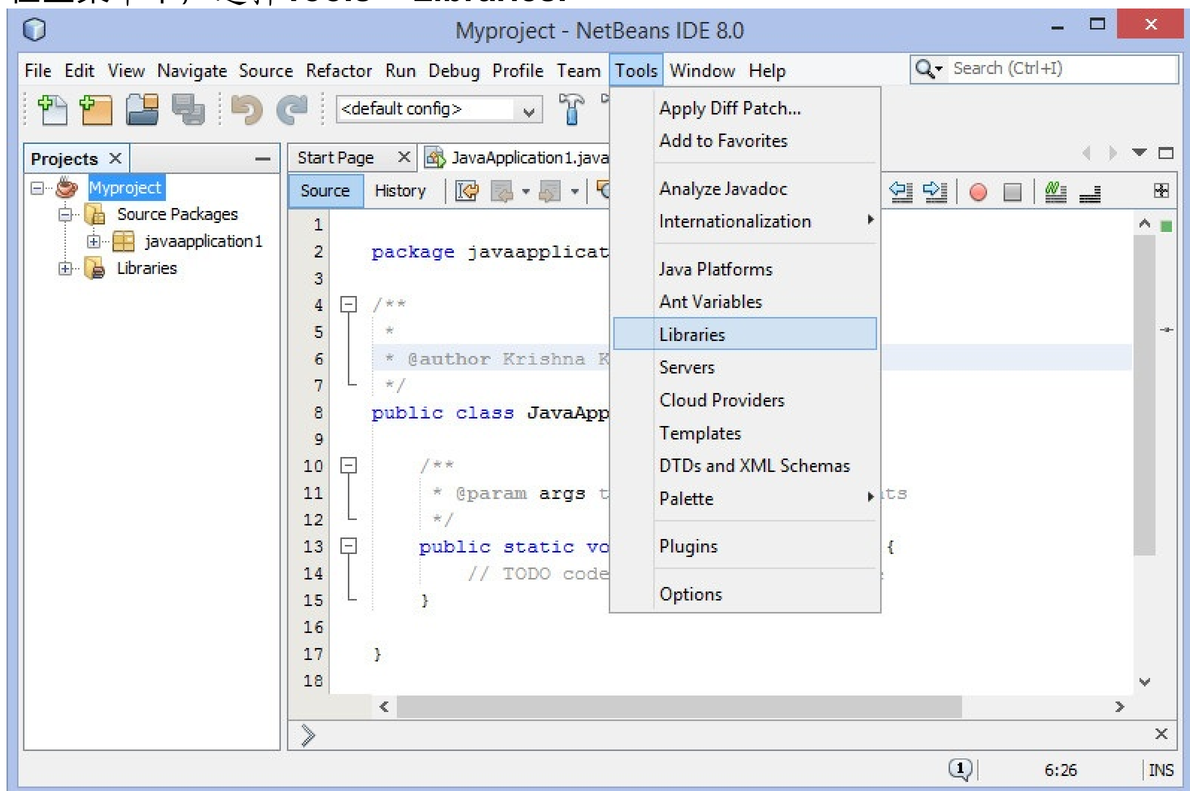
4. 点击 “Add External JARs…” 按钮.
5. 现在选择的路径，其中原生库文件 ('gluegen-rt-natives-windows-amd64.jar and joglall-natives-windows-amd64.jar') 位置.
6. 重复同样的程序源代码。
7. 上文两个本地库文件，我们可以设置为Javadoc，源代码和jar文件的位置以相同的方式在 glegen-rt.jar 和 glugen-natives-windows-amd64.jar.

# Netbeans4.4设置JOGL

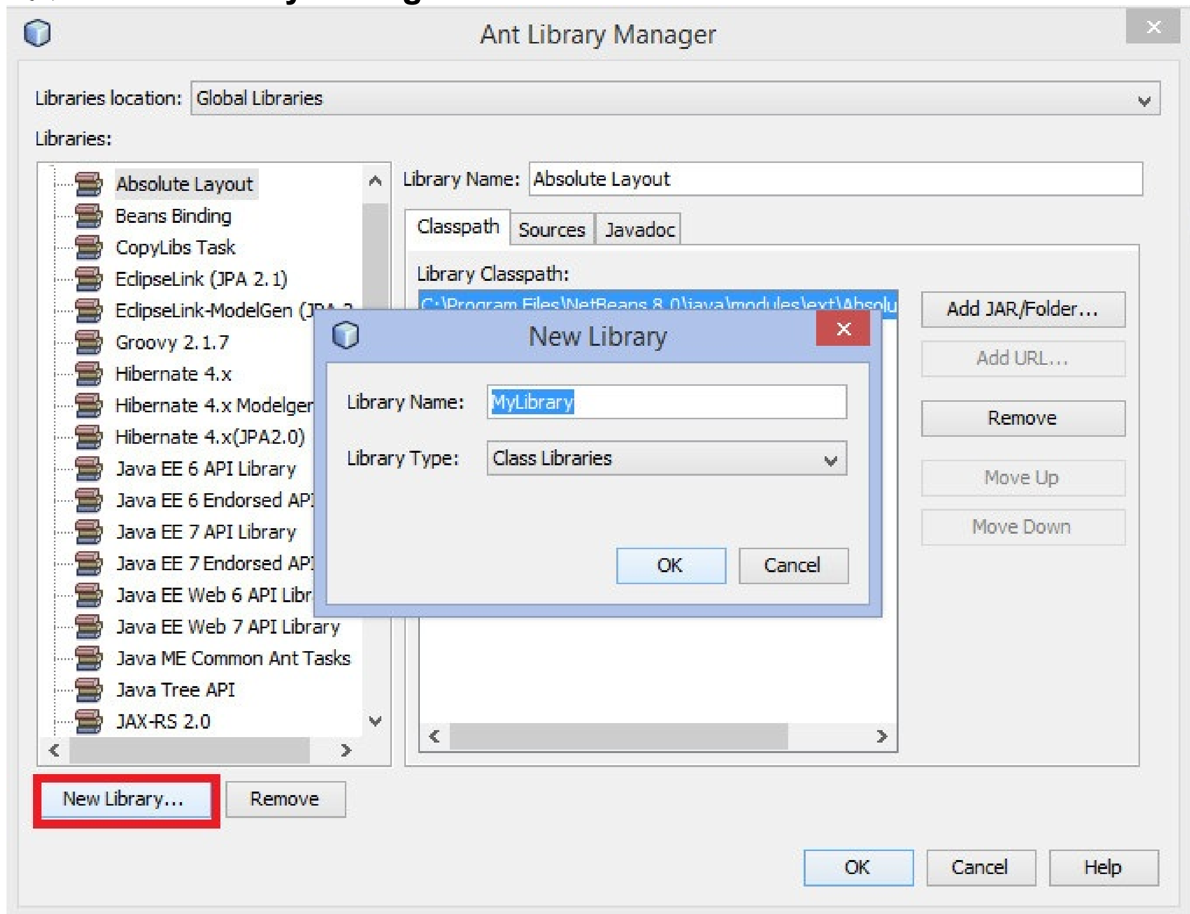
让我们通过以下步骤来设置JOGL针对NetBeans4.4：

## 添加库

1. 在主菜单中，选择**Tools > Libraries.**



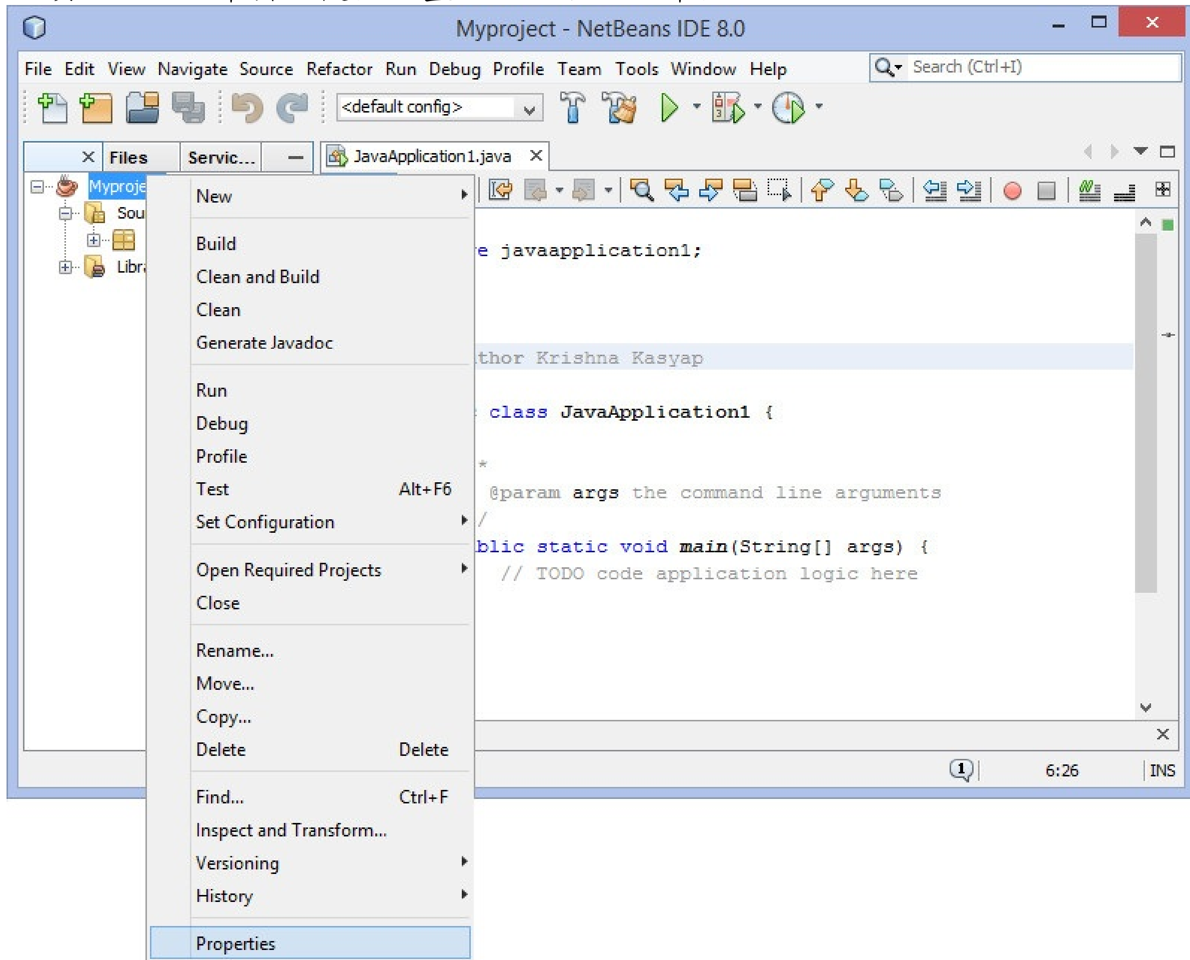
2. 这使得 **Ant Library Manager.**




3. 在Classpath选项卡，单击位于左下角新建库按钮。它会打开一个小对话框。
4. 输入库名称 JoGl2.0.
5. 点击 “OK” 按钮.
6. 点击 “Add JAR/Folder…” 按钮.
7. 选择在.jar文件jogl.all.jar和gluegen-rt.jar所在的路径。

包括JOGL库到每一个项目，请执行以下步骤：

1. 在项目名称上单击右键。它显示一个快捷菜单。




2. 选择属性。它打开了一个名为Project Properties窗口。
3. 从左侧的类别中选择库
4. 选择编译选项卡，并单击“添加库...”按钮。添加库对话框出现。
5. 现在添加JOGL2.0库，先前创建的。

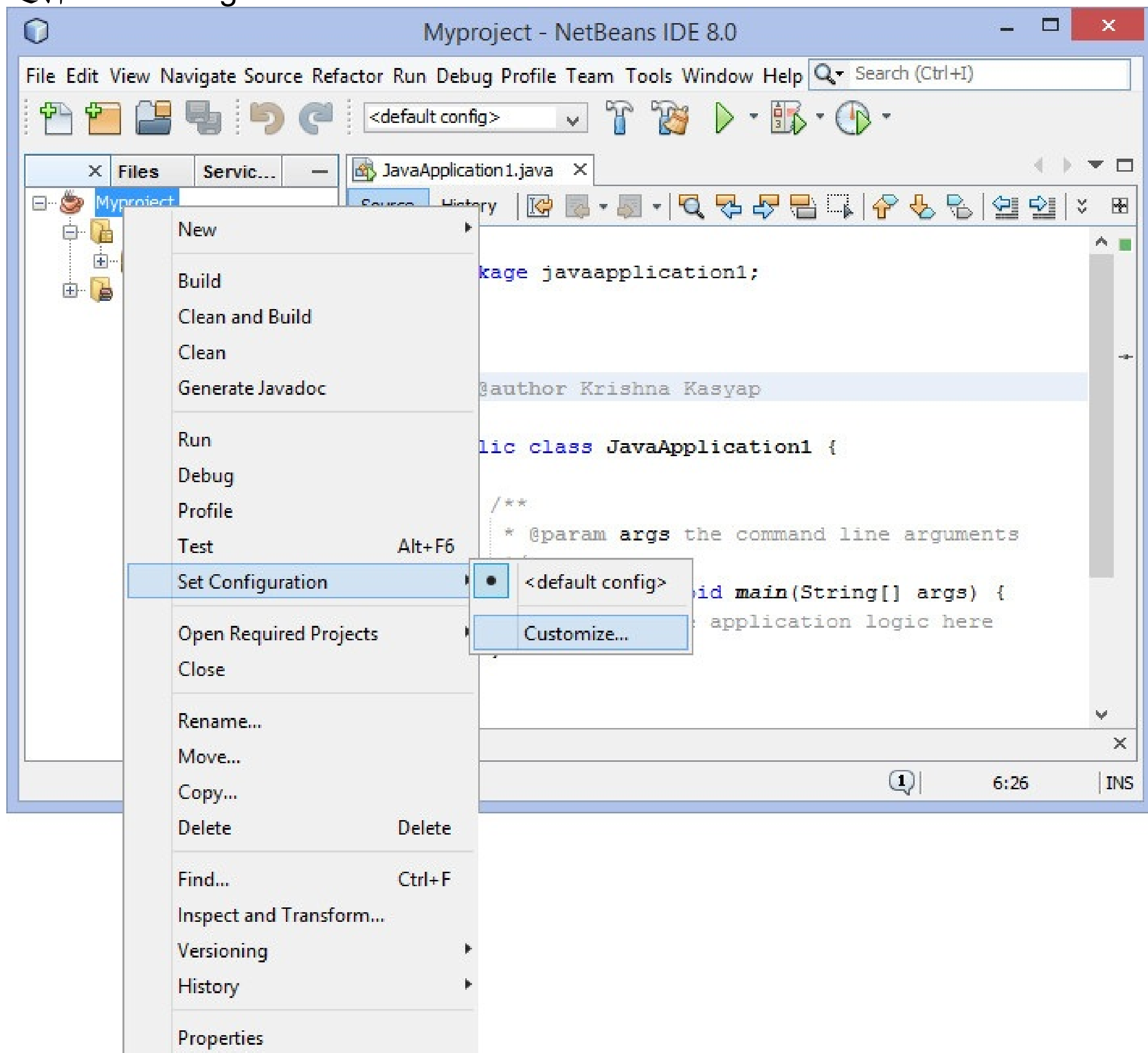
## 包括每个项目的本地库

按照给定的步骤，包括在每个项目中机库：

这使得项目属性窗口。

1. 用鼠标右键单击该项目。

2. 选择 Set Configuration > Customize…




3. 在右手边，在VM选项，点击“Customize”按钮。

4. 浏览包含JOGL本地库的路径 gluegen-rt-natives-windows-amd64.jar'' 和 'jogl-all-natives-windows-amd64.jar.

## 添加本地库Java文档

需要再次打开Ant库管理，使源代码和Javadoc可用于每个项目。按照给定的步骤：

## 添加本地库的源代码

1. 打开main menu.

2. 选择 Tools > Libraries. 这使 Library manager.

3. 在JavaDoc 下面选项卡中，单击 “New Library…” 按钮.

4. 输入 JOGLJavadoc 名称. (可以输入任何所需的名称)

5. 点击 “Add jars/libraries…” 按钮.

6. 其中，选择解压后的文件JOGL代码所在的路径。
7. 在Sources 下面的选项卡中，单击 “New Library…” 按钮。进入JOGLsources名称.
8. 点击 “Add jars/libraries…” 按钮，选择解压缩源代码所在的路径。

## 自定义JDK编辑

1. 设置类路径的文件jogl.all.jar和gluegen-rt.jar。
2. 设置路径，本地库gluegen-rt-natives-windows-amd64.jar 和 jogl-all-natives-windowsamd64.jar复制并将其粘贴到JSE lib下载文件夹中的所有jar文件。

# JOGL基本模板 - JOGL教程

本章介绍了编写JOGL基本模板的概念。

## 重要的接口和类

为了使程序能够使用JOGL图形API，需要实现GLEventListener接口。

### GLEventListener接口

可以在javax.media.opengl包找到GLEventListener接口。

Interface: GLEventListener

Package: javax.media.opengl

下表给出了各种方法和GLEventListener接口的详细描述：

| Sr. No. | 方法和说明 |
|---|---|
| 1 | **Void display(GLAutoDrawable drawable)**这就是所谓GLAutoDrawable接口的对象，由客户机发起OpenGL渲染。也就是说，该方法包含用于绘制使用OpenGL API的图形元素的逻辑。 |
| 2 | **Void dispose(GLAutoDrawable drawable)**这种方法的信号监听执行每各GLContext，所有的OpenGL释放资源，如内存缓冲区和GLSL程序。 |
| 3 | **Void init(GLAutoDrawble drawble)**这就是所谓GLAutoDrawable接口OpenGL上下文被初始化之后的对象。 |
| 4 | **Void reshape(GLAutoDrawble drawble,in tx,int y,int width ,int height)**第一重画过程中它被称为GLAutoDrawable接口的对象的组件大小后。它也被称为每当窗口上的部件的位置变化。 |

GLEventListener所有的方法都需要GLAutoDrawable接口作为参数的对象。

### GLAutoDrawable 接口

这个接口提供一个基于事件机制（GLEventListener），用于执行OpenGL渲染。GLAutoDrawable自动创建一个与GLAutoDrawable为对象的生命周期相关联的一个主要呈现上下文。

Interface: GLAutoDrawble

Package: javax.media.opengl

| Sr. No. | 方法和说明 |
|---|---|
| 1 | **GL getGL()**此方法返回所使用的GLAutoDrawable接口的当前对象的GL管道对象。 |
| 2 | **Void addGLEventListener(GLEventListener Listener)**这种方法增加了给定侦听器到当前绘制队列的末尾。 |
| 3 | **Void addGLEventListener(int index,GLEventListener listener)**这种方法增加了给定侦听器这个队列中绘制的给定索引处。 |
| 4 | **Void destroy()**这种方法会破坏GLAutoDrawable接口名为此对象，包括GLContext相关联的所有资源。 |

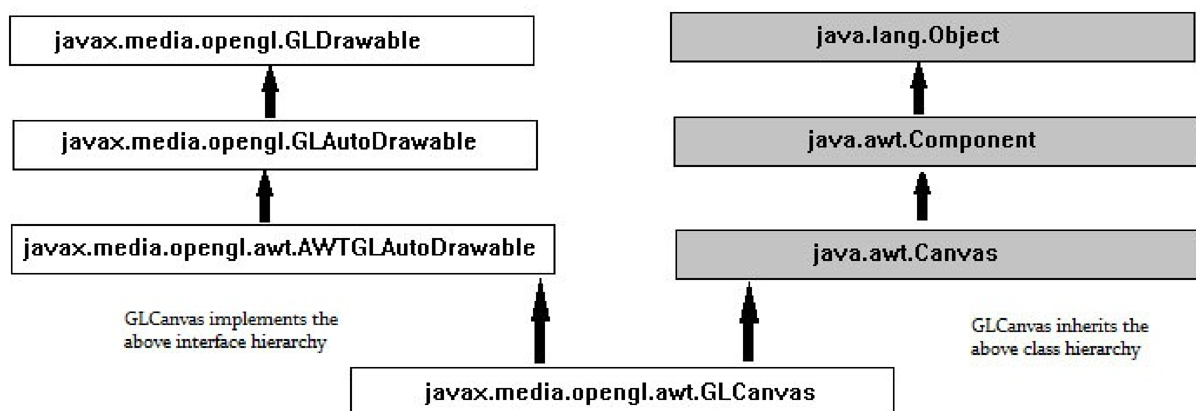
注：有在这个包的其他方法。只有很少的关于模板的重要方法是在此界面中进行讨论。

## GLCanvas 类

GLCanvas 和 GLJpanel 是JOGL GUI两大类实现 GLAutoDrawable 接口，这让他们可以用作拉丝表面的OpenGL命令。

GLCanvas是一个重量级AWT组件，它提供了OpenGL的渲染支持。这是一AWTAutoGLDrawable接口的主执行。它还是java.awt.Canvas继承类。因为它是一个重量级的组成部分，在某些情况下，GLJCanvas可能无法与Swing组件正确地结合起来。因此，必须谨慎使用，同时它与Swing共用。每当面临GLJCanvas问题，那么必须使用GLJPanel类。

GLCanvas类的分层图可以如下所示：




- GLEventistener接口工作以及GLCanvas类，它响应变化GLCanvas类并由它们制成要求的图纸。
- 每当GLCanvas的类被实例化，调用GLEventListener的init()方法。可以覆盖此方法来初始化OpenGL的状态。

- 每当GLCanvas的初始绘制（实例化），或调整大小，则执行GLEventListener的reshape()方法。它用来初始化OpenGL的视口和投影矩阵。它也被调用被改变来组件的位置。
- GLEventListener()方法显示包含渲染3D场景的代码。它被调用时调用GLCanvas的display()方法。

class: GLCanvas

package: javax.media.opengl.awt

Contructor

```
**GLCanvas()**
```

它与OpenGL功能的默认设置创建一个新的GLCanvas组成部分，使用默认的OpenGL功能选择机制，在默认的屏幕设备上。

```
**GLCanvas(GLCapabilitiesImmutable)**
```

它创建了一个新的GLCanvas成分与所请求的一套OpenGL的功能，使用默认的OpenGL功能选择机制，在默认的屏幕设备上。

| Sr. No. | 方法及描述 |
|---|---|
| 1 | **Void addGLEventListener(GLEventListener listener)**它增加了给定侦听器来绘制这个队列的末尾 |
| 2 | **Void addGLEventListener(int indexGLEventListener listener)**它增加了给定侦听器这个队列中绘制的给定索引处。 |

要实例化GLCanvas类，需要GLCapabilitiesImmutable接口，它指定一个不可变的集合的OpenGL性能的目的。

来获得 CapabilitiesImmutable 接口的对象的一个方式是实例 GLCapabilities 类，它实现接口。 GLCapabilities 类的实例可以用来服务。

## GLCapabilities 类

此类指定一组的OpenGL性能。它需要GLCapabilities对象作为参数。该GLCapabilities类描述渲染上下文必须支持所期望的功能，例如OpenGL的轮廓。

class: GLCapabilities

package: javax.media.opengl

Constructor

```
**GLCapabilities(GLProfile glprofile)**
```

创建一个GLCapabilities对象。

要实例化GLCanvas类，需要GLCapabilitiesImmutable接口，它指定一个不可变的集合的OpenGL性能的目的。

获得CapabilitiesImmutable接口的对象的一种方式是实例化GLCapabilities类并实现接口。 GLCapabilities类的实例，可以用来以服务为目的。

## GLCapabilities 类

此类指定一组OpenGL性能。它需要GLCapabilities对象作为参数。GLCapabilities类描述渲染上下文必须支持所期望的功能，例如OpenGL的轮廓。

class: GLCapabilities

package: javax.media.opengl

Constructor

```
**GLCapabilities(GLProfile glprofile)**
```

GLCapabilities类又需要GLProfile对象。

## GLProfile 类

由于几个版本的OpenGL API发布，需要指定OpenGL的API确切版本被用在你的程序到Java虚拟机（JVM）。这是通过使用GLProfile类。这个类的get()方法接受不同的预定义的String对象作为参数。每一个String对象是一个接口的名称，每个接口支持OpenGL的某些版本。如果初始化这个类的静态和单例，这个类提供了每个可用JOGL单例GLProfile对象。

class: GLProfile

package: javax.media.opengl

| Method and 描述 |
| --- |
| **Static GLProfile get(String profile)**使用默认设备。 |

因为这是一个静态方法，需要使用类名来调用它，它需要一个预定义的静态字符串变量作为参数。有在这12级这样的变量，分别代表GL接口的独立实现。

```
GLProfile.get(GLProfile.GL2);
```

下表显示GLProfile类的get()方法的字符串参数：

| Sr. No. | 预定义的字符串值（接口名称）和描述 |
|---|---|
| 1 | **GL2**这种接口包含所有的OpenGL[1.0 ...3.0]的方法，以及它的大多数扩展中定义在本说明书中的时间。 |
| 2 | **GLES1**这种接口包含所有的OpenGL ES[1.0 ...1.1]的方法，以及它的大多数扩展中定义在本说明书中的时间 |
| 3 | **GLES2**这种接口包含所有的OpenGL ES 2.0的方法，以及它的大部分在本说明书中的时间定义的扩展。 |
| 4 | **GLES3**这种接口包含所有的OpenGL ES3.0的方法，以及它的大部分在本说明书中的时间定义的扩展。 |
| 5 | **GL2ES1**此接口包含GL2和GLES1的公共子集。 |
| 6 | **GL2ES2**此接口包含GL3，GL2和GLES2的公共子集。 |
| 7 | **GL2GL3**此接口包含核心GL3（OpenGL的3.1+）和GL2的公共子集。 |
| 8 | **GL3**这种接口包含所有的OpenGL[3.1...3.3]核心方法，以及它的大部分在本说明书中的时间定义的扩展。 |
| 9 | **GL3bc**这种接口包含所有的OpenGL[3.1...3.3]的相容性的方法，以及它的大部分在本说明书中的时间定义的扩展。 |
| 10 | **GL3ES3**接口含有核心GL3（OpenGL的3.1+）和GLES3（OpenGL ES3.0）的公共子集。 |
| 11 | **GL4**这种接口包含所有的OpenGL[4.0...4.3]核心方法，以及它的大部分在本说明书中的时间定义的扩展。 |
| 12 | **GL4bc**这种接口包含所有的OpenGL[4.0...4.3]相容性分布，以及它的大部分在本说明书中的时间定义的扩展。 |
| 13 | **GL4ES3**接口含有核心GL4（OpenGL的4.0+）和GLES3（OpenGL ES3.0）的公共子集。 |

现在一切都被设置在使用JOGL的第一个程序中。

## 使用画布与AWT的基本模板

使用JOGL的编程，可以绘制各种图形形状，例如直线，三角形，三维形状，包括特殊效果，如旋转，照明，色彩等。

JOGL编程的基本模板如下：

步骤1：创建一个类

最初创建一个实现 GlEventListener 接口的类，并导入包 javax.media.opengl。实现所有四种方法 display(), dispose(), reshape(), init()。由于这是基本框架，基本任务，如创建Canvas类，将其添加到框架进行了讨论。所有GLEVentListener接口的方法留下未实现。

第二步：准备画布

(a)构建GLCanvas类和对象

```
final GLCanvas glcanvas = new GLCanvas( xxxxxxx );

  //here capabilities obj should be passed as parameter
```

(b) 实例化GLCapabilities类

```
GLCapabilities capabilities = new GLCapabilities( xxxxx );

   //here profile obj should be passed as parameter
```

生成GLProfile对象

因为这是静态方法，它是使用类名调用。由于本教程是关于JOGL2，产生GL2接口对象。

```
final GLProfile profile = GLProfile.get( GLProfile.GL2 );
// both, variable and method are static hence both are called using
```

让我们看到了代码片段画布

```
//getting the capabilities object of GL2 profile
final GLProfile profile = GLProfile.get(GLProfile.GL2);
GLCapabilities capabilities = new GLCapabilities(profile);
// The canvas
final GLCanvas glcanvas = new GLCanvas(capabilities);
```

现在，使用方法addGLEventListener()添加GLEventListener到画布。此方法需要GLEventListener接口参数的对象。因此，通过实现GLEventListener类的对象。

```
BasicFrame basicframe=newBasic Frame( );// class which implements
GLEventListener interface
glcanvas.addGLEventListener( basicframe );
```

使用setSize()方法继承自javax.media.opengl.awt.AWTGLAutoDrawable GLCanvas框架设置大小。

```
glcanvas.setSize( 400, 400 );
```

现在，还可以用GLCanvas。

第三步：创建框架

通过实例JSE AWT框架组件的Frame类对象创建的框架。

添加画布，使框架可见。

```
//creating frame
final Frame frame = new frame( " Basic Frame" );
//adding canvas to frame
frame.add( glcanvas );
frame.setVisible( true );
```

第**4**步：在全屏观看帧

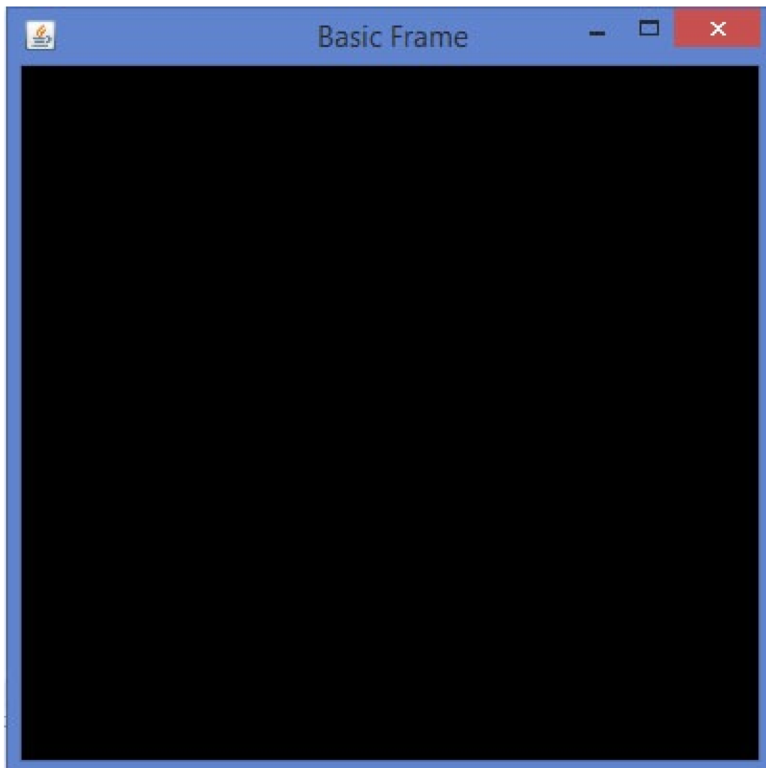
对于帧的全屏视图，使用java.awt.Toolkit中的类得到默认的屏幕大小。现在，使用这些默认的屏幕大小尺寸，使用setSize()方法设置帧的大小。

```
Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
frame.setSize(screenSize.width, screenSize.height);
```

让我们通过在程序使用AWT来生成基本的框架：

```
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;
public class BasicFrame implements GLEventListener{
   @Override
   public void display(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void dispose(GLAutoDrawable arg0) {
      //method body
   }
   @Override
   public void init(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void reshape(GLAutoDrawable arg0, int arg1, int arg2, int
      // method body
   }
   public static void main(String[] args) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get(GLProfile.GL2);
      GLCapabilities capabilities = new GLCapabilities(profile);
      // The canvas
      final GLCanvas glcanvas = new GLCanvas(capabilities);
      BasicFrame b = new BasicFrame();
      glcanvas.addGLEventListener(b);
      glcanvas.setSize(400, 400);
      //creating frame
      final JFrame frame = new JFrame (" Basic Frame");
      //adding canvas to frame
      frame.getContentPane().add(glcanvas);
      frame.setSize(frame.getContentPane().getPreferredSize());
      frame.setVisible(true);
   }
}
```

如果编译并执行上述程序，将生成以下输出。它显示了当我们用GLCanvas类与AWT形成一个基本的框架：




## 使用画布与**Swing**

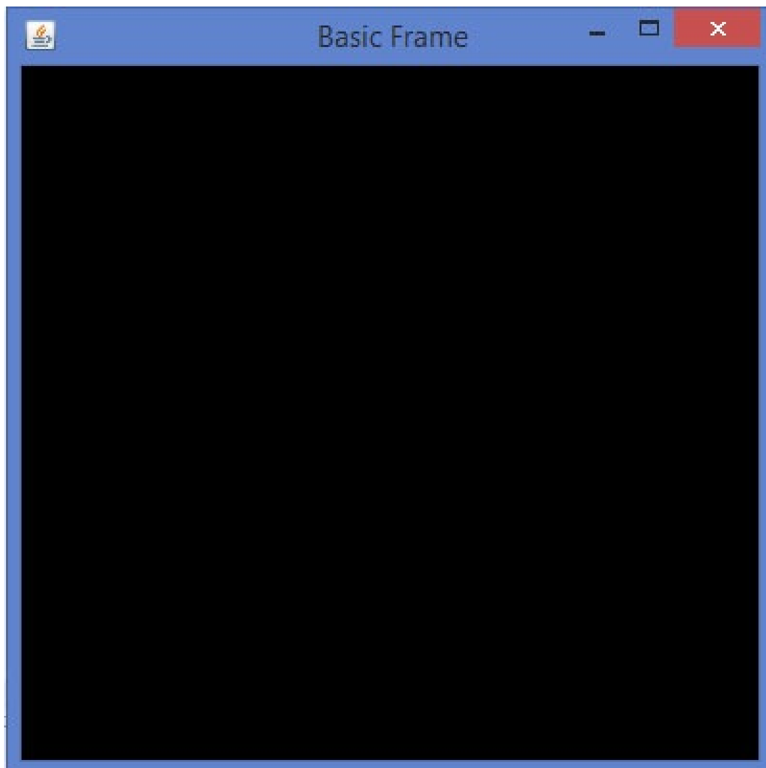
使用Canvas与AWT提供了一个图形化的框架与重量级的功能。对于具有轻量级的图形框架，同时采用GLCanvas与Swing，可以将GLCanvas会在窗口JFrame直接使用，也可以将其添加到JPanel中。

下面的程序将生成使用GLCanvas与Swing窗口的基本框架：

```
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class BasicFrame implements GLEventListener{
   @Override
   public void display(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void dispose(GLAutoDrawable arg0) {
      //method body
   }
   @Override
   public void init(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void reshape(GLAutoDrawable arg0, int arg1, int arg2, int
      // method body
   }
   public static void main(String[] args) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get(GLProfile.GL2);
      GLCapabilities capabilities = new GLCapabilities(profile);
      // The canvas
      final GLCanvas glcanvas = new GLCanvas(capabilities);
      BasicFrame b = new BasicFrame();
      glcanvas.addGLEventListener(b);
      glcanvas.setSize(400, 400);
      //creating frame
      final JFrame frame = new JFrame (" Basic Frame");
      //adding canvas to it
      frame.getContentPane().add(glcanvas);
      frame.setSize(frame.getContentPane().getPreferredSize());
      frame.setVisible(true);
   }//end of main
}//end of classimport
```

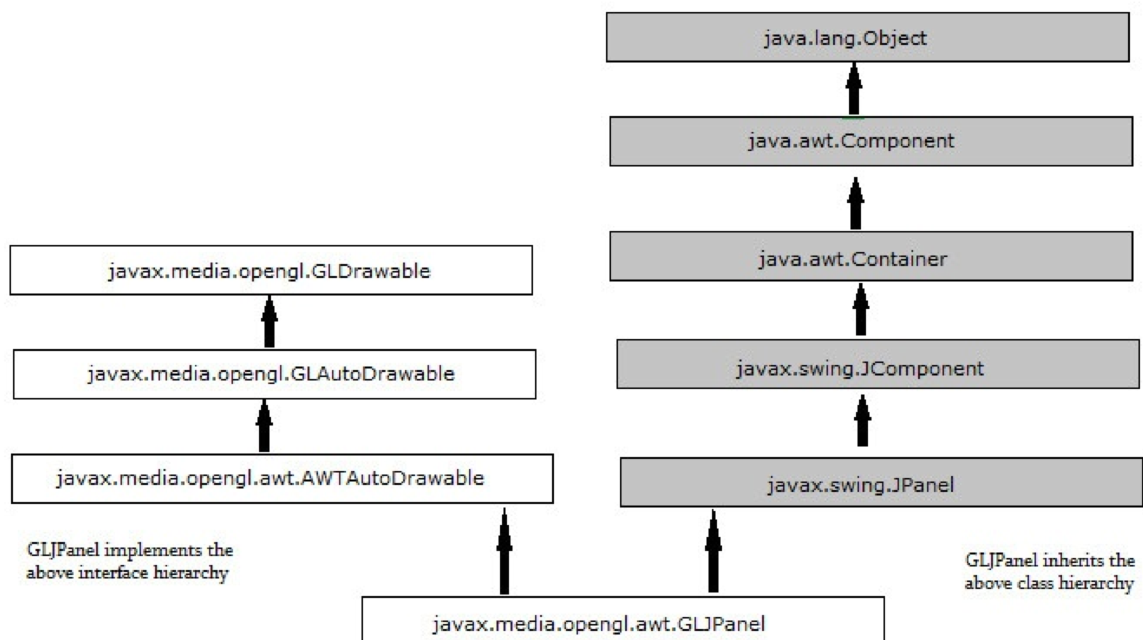
如果编译并执行上述程序，将生成以下输出。它显示了当我们用GLCanvas与Swing窗口形成一个基本的框架。




## GLJPanel 类

它是一个轻量级的Swing组件，它提供了OpenGL的渲染支持。它提供了与Swing的兼容性。

GLJPanel类层次结构




class: GLJPanel

package: javax.media.opengl.awt

Contructors

```
GJPanel()
```

创建一个用 OpenGL 功能的默认设置一个新的GLJPanel组成部分。

```
(GLCapabilitiesImmutable)
```

创建一个具有指定一组OpenGL性能的新 GLJPanel 组件。

```
GLJPanel(GLCapabilitiesImmutable userCapsRequest,
GLCapabilitiesChooser chooser)
```

创建一个新的GLJPanel组件。

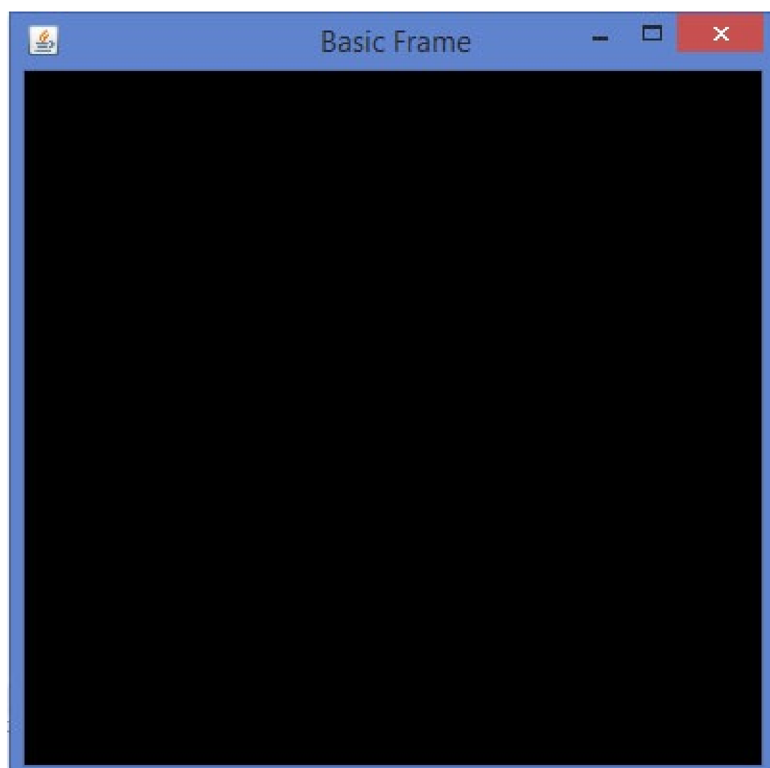
| 方法及描述 |
| --- |
| **Void addGLEventListener(GLEventListener listener)**此方法将给定的侦听器，以这种绘制队列的末尾。 |
| **Void addGLEventListener(int indexGLEventListener listener)**这种方法绘制该队列的特定索引处添加指定的侦听器。 |

# 使用GLJPanel与Swing窗口

让我们来看看生成使用GLJPanel与Swing窗口基本框架方案：

```
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;
public class BasicFrame implements GLEventListener{
   @Override
   public void display(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void dispose(GLAutoDrawable arg0) {
      //method body
   }
   @Override
   public void init(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void reshape(GLAutoDrawable arg0, int arg1, int arg2, int
      int arg4) {
      // method body
   }
   public static void main(String[] args) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get(GLProfile.GL2);
      GLCapabilities capabilities = new GLCapabilities(profile);
      // The canvas
      final GLCanvas glcanvas = new GLCanvas(capabilities);
      BasicFrame b = new BasicFrame();
      glcanvas.addGLEventListener(b);
      glcanvas.setSize(400, 400);
      //creating frame
      final JFrame frame = new JFrame (" Basic Frame");
      //adding canvas to it
      frame.getContentPane().add(glcanvas);
      frame.setSize(frame.getContentPane().getPreferredSize());
      frame.setVisible(true);
   }//end of main
}//end of classimport
```

如果编译并执行上述程序，将生成以下输出。这表明，当我们使用GLJPanel swing窗口形成一个基本框架：

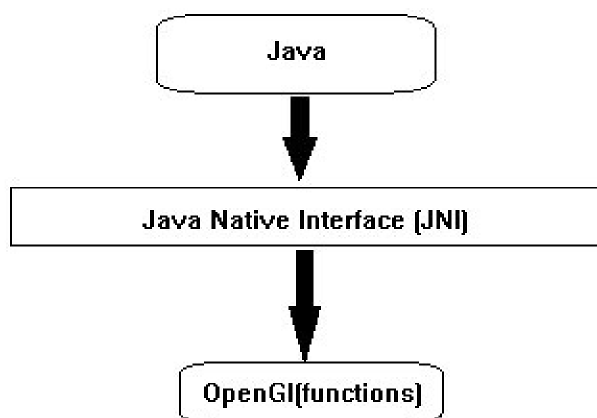

Basic Frame

# JOGL图形形状 - JOGL教程

本教程介绍了绘制直线，用直线的各种形状。 OpenGL的API提供了原始的方法，用这些方法，可以开发形状，如三角形，多边形和圆形绘制基本图形元素如点，顶点，线等。或者二维和三维。

## 图形对象

要访问程序特定于硬件和操作系统平台，以及其他语言编写，比如C和C++(原生应用)库，Java使用一种称为Java本地接口(JNI)编程框架的工作。 JOGL内部使用此接口，如图中下面的图表来访问OpenGL函数。




GLEventListener接口的所有四种方法让代码(Java JOGL方法)，它内部调用OpenGL函数，这些JOGL方法的命名也类似于 OpenGL 命名约定。如果在OpenGL中的函数名是在glBegin()，它被用作gl.glBegin()。

只要gl.glBegin()的Java JOGL的方法被调用时，它在内部调用OpenGL的glBegin()方法。这是在安装JOGL的时间对用户的系统上安装本地库文件的原因。

Display() 方法

这是其中包含用于开发图形的代码的一个重要方法。这就要求GLAutoDrawable接口对象作为参数。

Display()方法中，首先得到使用GL接口的对象的OpenGL上下文（GL继承GLBase接口，该接口包含的方法来生成所有的OpenGL上下文对象）。由于本教程是关于JOGL2让我们产生GL2对象。

让我们通过代码片段获取GL2对象：

```
//Generating GL object
GL gl=drawable.getGL();
GL gl=drawable.getGL();
//Using this Getting the Gl2 Object
//this can be written in a single line like
final GL2 gl = drawable.getGL().getGL2();
```

使用GL2接口的对象，就可以访问GL2接口的成员，而这又提供了访问OpenGL[1.0 ...3.0]功能。

## 绘制一条线

GL2接口包含的方法和列表，但这里的三个主要方法的重要论述，即函数是glBegin()glVertex()和glEnd()。

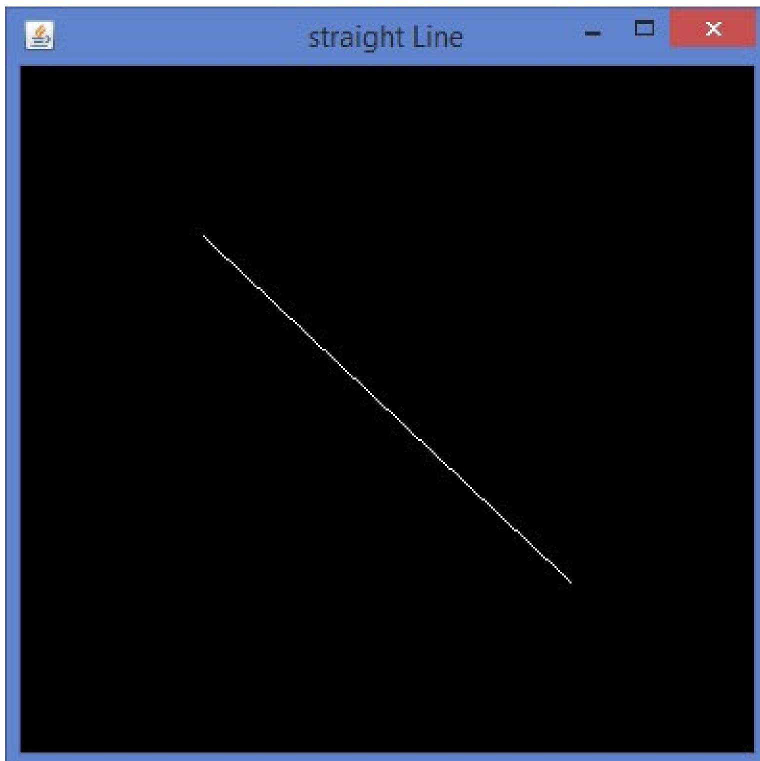
| Sr. No. | 方法及描述 |
|---|---|
| 1 | **glBegin()**此方法开始画线过程。它采用预定义的字符串整数“GL_LINES”作为一个参数，它是由GL接口继承。 |
| 2 | **glVertex3f()/glVertex2f()**此方法创建的顶点，我们必须通过坐标参数3f和 2f,，这表示3维的浮点坐标和2维浮点分别坐标。 |
| 3 | **glEnd()**行结尾 |

让我们通过程序来绘制一条直线：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class Line implements GLEventListener{
   @Override
   public void display(GLAutoDrawable drawable) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glBegin (GL2.GL_LINES);//static field
      gl.glVertex3f(0.50f,-0.50f,0);
      gl.glVertex3f(-0.50f,0.50f,0);
      gl.glEnd();
   }
   @Override
   public void dispose(GLAutoDrawable arg0) {
      //method body
   }

   @Override
   public void init(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void reshape(GLAutoDrawable arg0, int arg1, int arg2, int
      // method body
   }
   public static void main(String[] args) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get(GLProfile.GL2);
      GLCapabilities capabilities = new GLCapabilities(profile);
      // The canvas
      final GLCanvas glcanvas = new GLCanvas(capabilities);
      Line l = new Line();
      glcanvas.addGLEventListener(l);
      glcanvas.setSize(400, 400);
      //creating frame
      final JFrame frame = new JFrame ("straight Line");
      //adding canvas to frame
      frame.getContentPane().add(glcanvas);
      frame.setSize(frame.getContentPane().getPreferredSize());
      frame.setVisible(true);
   }//end of main
}//end of classimport javax.media.opengl.GL2;
```




## 使用GL_LINES绘制形状

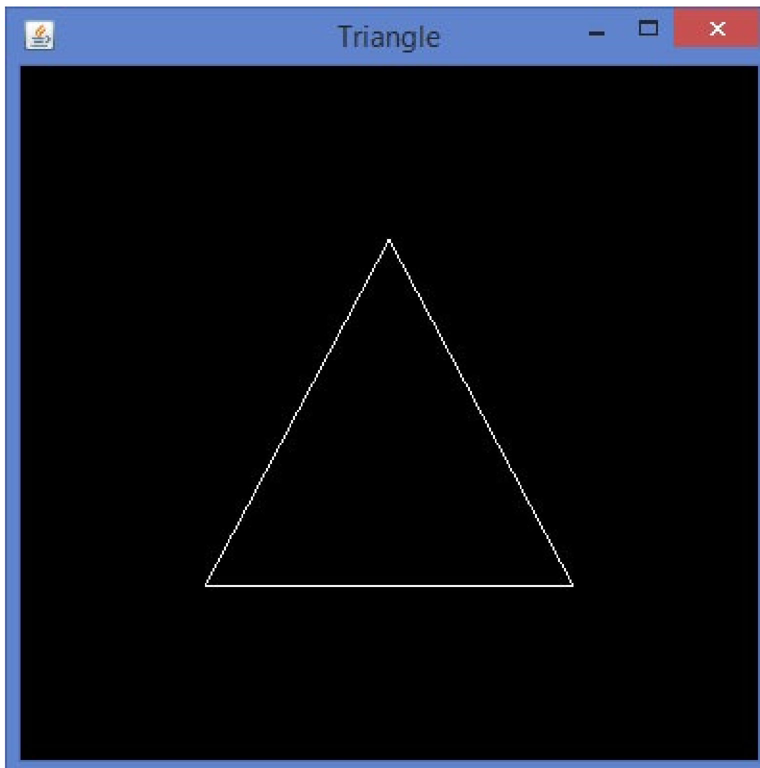
让我们通过一个程序使用GL_LINES绘制一个三角形：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class Triangle implements GLEventListener{
   @Override
   public void display(GLAutoDrawable drawable) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glBegin (GL2.GL_LINES);
      //drawing the base
      gl.glBegin (GL2.GL_LINES);
      gl.glVertex3f(-0.50f, -0.50f, 0);
      gl.glVertex3f(0.50f, -0.50f, 0);
      gl.glEnd();
      //drawing the right edge
      gl.glBegin (GL2.GL_LINES);
      gl.glVertex3f(0f, 0.50f, 0);
      gl.glVertex3f(-0.50f, -0.50f, 0);
      gl.glEnd();
      //drawing the lft edge
      gl.glBegin (GL2.GL_LINES);
```

```
        gl.glVertex3f(0f, 0.50f, 0);
        gl.glVertex3f(0.50f, -0.50f, 0);
        gl.glEnd();
        gl.glFlush();
     }
     @Override
     public void dispose(GLAutoDrawable arg0) {
        //method body
     }
     @Override
     public void init(GLAutoDrawable arg0) {
     // method body
     }
     @Override
     public void reshape(GLAutoDrawable arg0, int arg1, int arg2, int
     int arg4) {
     // method body
     }
     public static void main(String[] args) {
        //getting the capabilities object of GL2 profile
        final GLProfile profile = GLProfile.get(GLProfile.GL2);
        GLCapabilities capabilities = new GLCapabilities(profile);
        // The canvas
        final GLCanvas glcanvas = new GLCanvas(capabilities);
        Triangle l = new Triangle();
        glcanvas.addGLEventListener(l);
        glcanvas.setSize(400, 400);
        //creating frame
        final JFrame frame = new JFrame ("Triangle");
        //adding canvas to frame
        frame.getContentPane().add(glcanvas);
        frame.setSize(frame.getContentPane().getPreferredSize());
        frame.setVisible(true);
     }//end of main
  }//end of classimport javax.media.opengl.GL2;
```

如果编译并执行上述程序，将生成以下输出。它示出了使用glBegin()方法的GL_LINES画出一个三角形。



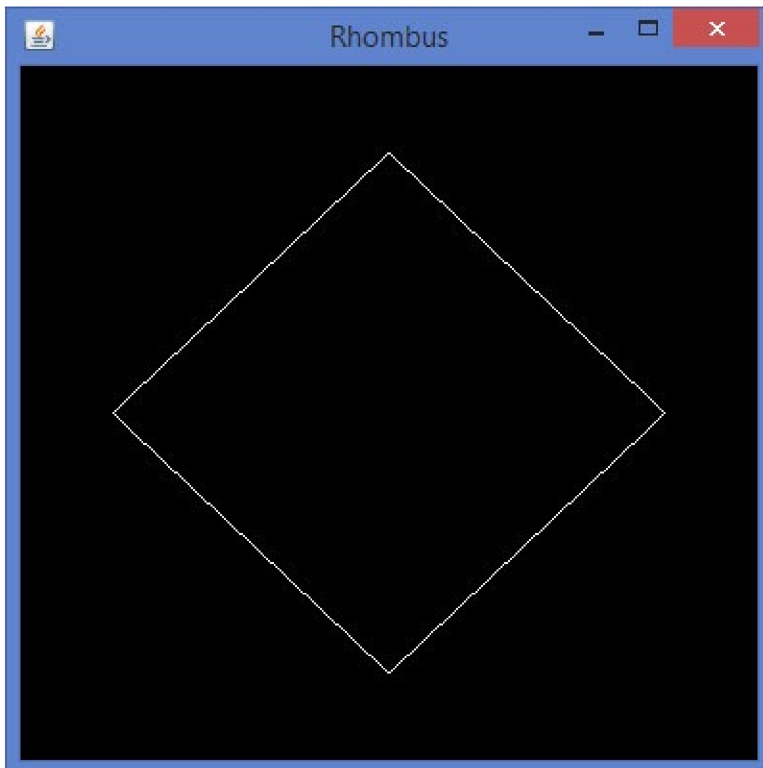

让我们通过一个程序中使用GL_LINES画一个菱形：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class Rhombus implements GLEventListener{
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      //edge1
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( 0.0f,0.75f,0 );
      gl.glVertex3f( -0.75f,0f,0 );
      gl.glEnd();
      //edge2
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( -0.75f,0f,0 );
      gl.glVertex3f( 0f,-0.75f, 0 );
      gl.glEnd();
      //edge3
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( 0f,-0.75f, 0 );
      gl.glVertex3f( 0.75f,0f, 0 );
      gl.glEnd();
      //edge4
```

```
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( 0.75f,0f, 0 );
      gl.glVertex3f( 0.0f,0.75f,0 );
      gl.glEnd();
      gl.glFlush();
   }
   @Override
   public void dispose( GLAutoDrawable arg0 ) {
      //method body
   }
   @Override
   public void init(GLAutoDrawable arg0 ) {
      // method body
   }
   @Override
   public void reshape( GLAutoDrawable arg0, int arg1, int arg2, ir
      // method body
   }
   public static void main( String[] args ) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get( GLProfile.GL2 );
      GLCapabilities capabilities = new GLCapabilities( profile );
      // The canvas
      final GLCanvas glcanvas = new GLCanvas( capabilities );
      Rhombus rhombus = new Rhombus();
      glcanvas.addGLEventListener( rhombus );
      glcanvas.setSize( 400, 400 );
      //creating frame
      final JFrame frame = new JFrame ( "Rhombus" );
      //adding canvas to frame
      frame.getContentPane().add( glcanvas );
      frame.setSize( frame.getContentPane().getPreferredSize() );
      frame.setVisible( true );
   }//end of main
}//end of class
```

如果编译并执行以上程序，会得到下面的输出。它示出了使用在glBegin()方法的GL_LINES产生一个菱形。




让我们通过一个程序使用GL_LINES画一所房子：

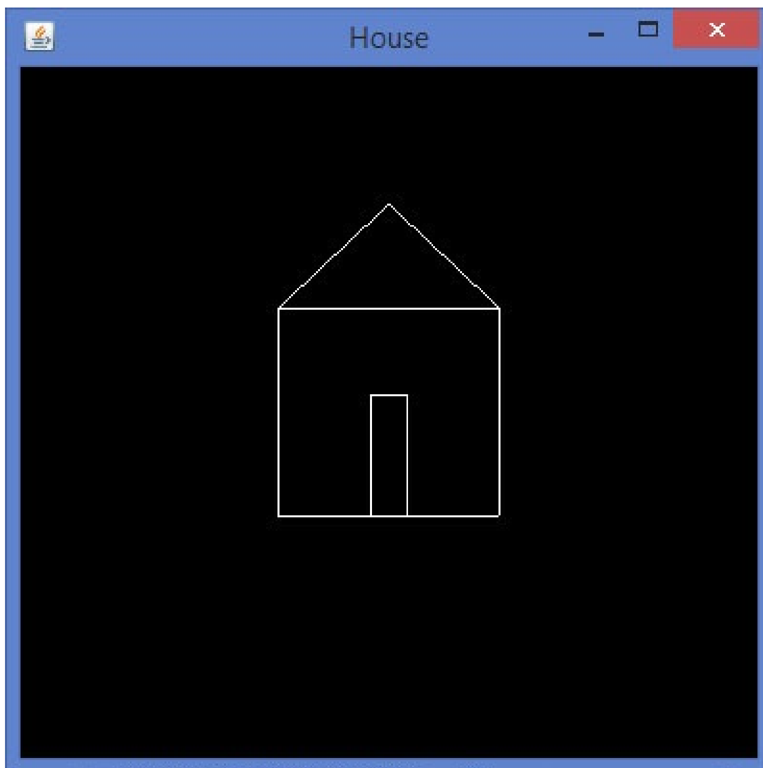
```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class House implements GLEventListener{
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      //drawing top
      gl.glBegin ( GL2.GL_LINES );
      gl.glVertex3f( -0.3f, 0.3f, 0 );
      gl.glVertex3f( 0.3f,0.3f, 0 );
      gl.glEnd();
      //drawing bottom
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( -0.3f,-0.3f, 0 );
      gl.glVertex3f( 0.3f,-0.3f, 0 );
      gl.glEnd();
      //drawing the right edge
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( -0.3f,0.3f, 0 );
      gl.glVertex3f( -0.3f,-0.3f, 0 );
      gl.glEnd();
      //drawing the left edge
```

```
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( 0.3f,0.3f,0 );
      gl.glVertex3f( 0.3f,-0.3f,0 );
      gl.glEnd();
      //building roof
      //building lft dia
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( 0f,0.6f, 0 );
      gl.glVertex3f( -0.3f,0.3f, 0 );
      gl.glEnd();
      //building rt  dia
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( 0f,0.6f, 0 );
      gl.glVertex3f( 0.3f,0.3f, 0 );
      gl.glEnd();
      //building door
      //drawing top
      gl.glBegin ( GL2.GL_LINES );
      gl.glVertex3f( -0.05f, 0.05f, 0 );
      gl.glVertex3f( 0.05f, 0.05f, 0 );
      gl.glEnd();
      //drawing the left edge
      gl.glBegin ( GL2.GL_LINES );
      gl.glVertex3f( -0.05f, 0.05f, 0 );
      gl.glVertex3f( -0.05f, -0.3f, 0 );
      gl.glEnd();
      //drawing the right edge
      gl.glBegin ( GL2.GL_LINES );
      gl.glVertex3f( 0.05f, 0.05f, 0 );
      gl.glVertex3f( 0.05f, -0.3f, 0 );
      gl.glEnd();
   }
   @Override
   public void dispose( GLAutoDrawable arg0 ) {
      //method body
   }
   @Override
   public void init( GLAutoDrawable arg0 ) {
      // method body
   }
   @Override
   public void reshape( GLAutoDrawable arg0, int arg1, int arg2, in
      // method body
   }
   public static void main( String[] args ) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get( GLProfile.GL2 );
      GLCapabilities capabilities = new GLCapabilities( profile );
      // The canvas
      final GLCanvas glcanvas = new GLCanvas( capabilities );
      House house = new House();
      glcanvas.addGLEventListener( house );
      glcanvas.setSize(400, 400);
```

```
        //creating frame
        final JFrame frame = new JFrame( "House" );
        //adding canvas to frame
        frame.getContentPane().add( glcanvas );
        frame.setSize( frame.getContentPane().getPreferredSize() );
        frame.setVisible( true );
    }//end of main
}//end of class
```

如果编译并执行以上程序，会得到下面的输出。它示出了使用GL_LINES()方法生成一所房子的图。




# 使用glBegin()更多的参数绘画出更多的形状

除了GL_LINES预定义的字符串参数，glBegin()方法接受八个参数。可以用它来绘制不同的形状。这些用于相同GL_LINES。

下表显示了glBegin()方法的参数和描述：

| Sr. No. | 参数和描述 |
|---|---|
| 1 | GL_LINES创建每对顶点作为一个独立的线段。 |
| 2 | GL_LINE_STRIP绘制线段的连接组从第一顶点到最后。 |
| 3 | GL_LINE_LOOP绘制线段的从第一顶点到最后一个连接组再次回到第一个点。 |
| 4 | GL_TRIANGLES把顶点的每一三元组作为一个独立的三角形。 |
| 5 | GL_TRIANGLE_STRIP绘制三角形的连接组。一个三角形被定义为所述第一两个顶点后呈现的每个顶点。 |
| 6 | GL_TRIANGLE_FAN绘制三角形的连接组。一个三角形被定义为所述第一两个顶点后呈现的每个顶点。 |
| 7 | GL_QUADS将每个组的四个顶点作为一个独立的四边形。 |
| 8 | GL_QUAD_STRIP绘制四边形的连接组。一个四边形被定义为每对所述第一对后呈现的顶点。 |
| 9 | GL_POLYGON绘制一个单一的，凸多边形。顶点1，...，N定义这个多边形。 |

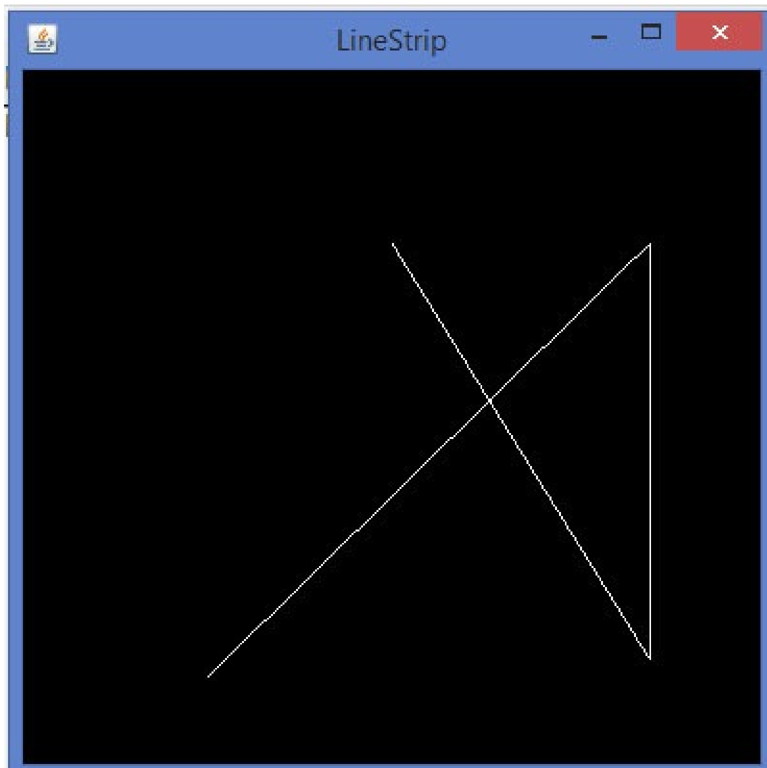
让我们来看看使用glBegin()参数的一些例子。

程序画线带钢：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class LineStrip implements GLEventListener{
   @Override
   public void display(GLAutoDrawable drawable) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glBegin (GL2.GL_LINE_STRIP);
      gl.glVertex3f(-0.50f,-0.75f, 0);
      gl.glVertex3f(0.7f,0.5f, 0);
      gl.glVertex3f(0.70f,-0.70f, 0);
      gl.glVertex3f(0f,0.5f, 0);
      gl.glEnd();
   }
   @Override
   public void dispose(GLAutoDrawable arg0) {
      //method body
   }
   @Override
   public void init(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void reshape(GLAutoDrawable arg0, int arg1, int arg2, int
      // method body
   }
   public static void main(String[] args) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get(GLProfile.GL2);
      GLCapabilities capabilities = new GLCapabilities(profile);
      // The canvas
      final GLCanvas glcanvas = new GLCanvas(capabilities);
      LineStrip r = new LineStrip();
      glcanvas.addGLEventListener(r);
      glcanvas.setSize(400, 400);
      //creating frame
      final JFrame frame = new JFrame ("LineStrip");
      //adding canvas to frame
      frame.getContentPane().add(glcanvas);
      frame.setSize(frame.getContentPane().getPreferredSize());
      frame.setVisible(true);
   }//end of main
}//end of classimport javax.media.opengl.GL2;
```

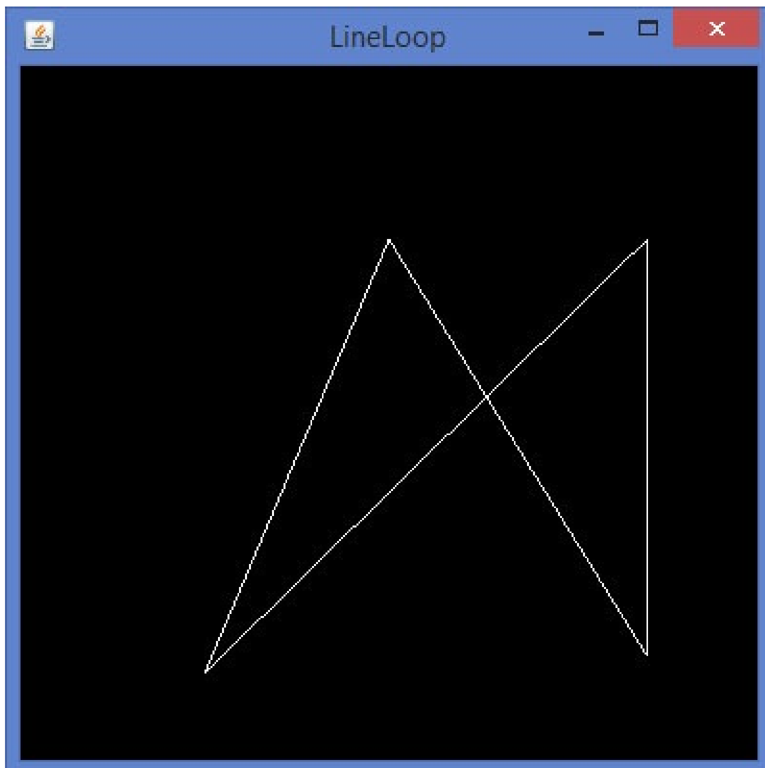
如果编译并执行上面的代码，生成以下输出：




代码片段display()方法来绘制线路回路：

```
public void display(GLAutoDrawable drawable) {
   final GL2 gl = drawable.getGL().getGL2();
   gl.glBegin (GL2.GL_LINE_LOOP);
   gl.glVertex3f( -0.50f, -0.75f, 0);
   gl.glVertex3f(0.7f, .5f, 0);
   gl.glVertex3f(0.70f,  -0.70f, 0);
   gl.glVertex3f(0f, 0.5f, 0);
   gl.glEnd();
}
```

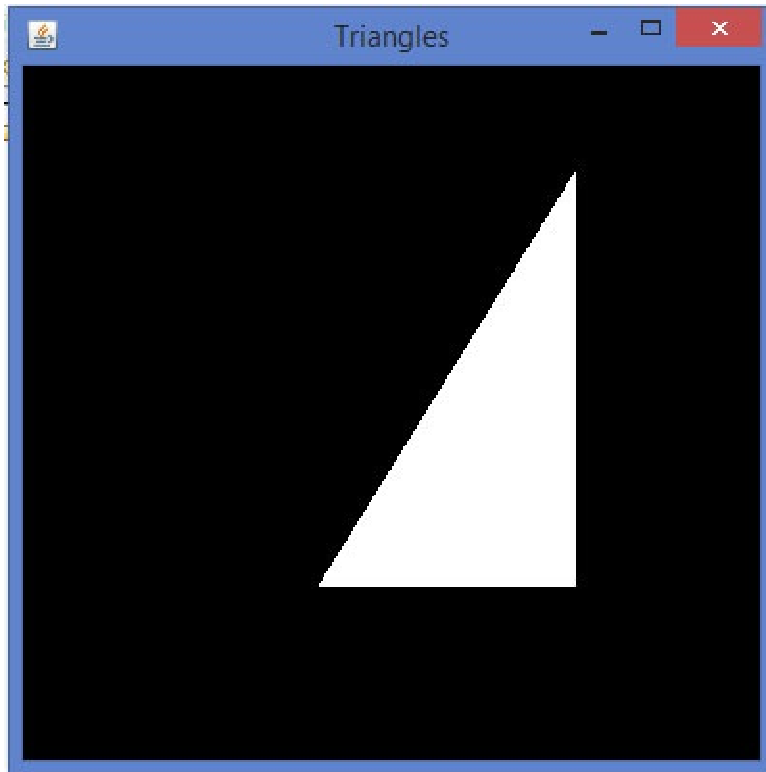
如果用上面的代码替换任何基本的模板方案的display()方法，编译并执行它，下面的输出生成：




代码片段display()方法使用GL_TRIANGLES画三角形

```
public void display(GLAutoDrawable drawable) {
   final GL2 gl = drawable.getGL().getGL2();
   gl.glBegin(GL2.GL_TRIANGLES);        // Drawing Using Triangles
   gl.glVertex3f(0.5f,0.7f,0.0f);       // Top
   gl.glVertex3f(-0.2f,-0.50f,0.0f);   // Bottom Left
   gl.glVertex3f(0.5f,-0.5f,0.0f);    //Bottom Right
   gl.glEnd();
}
```

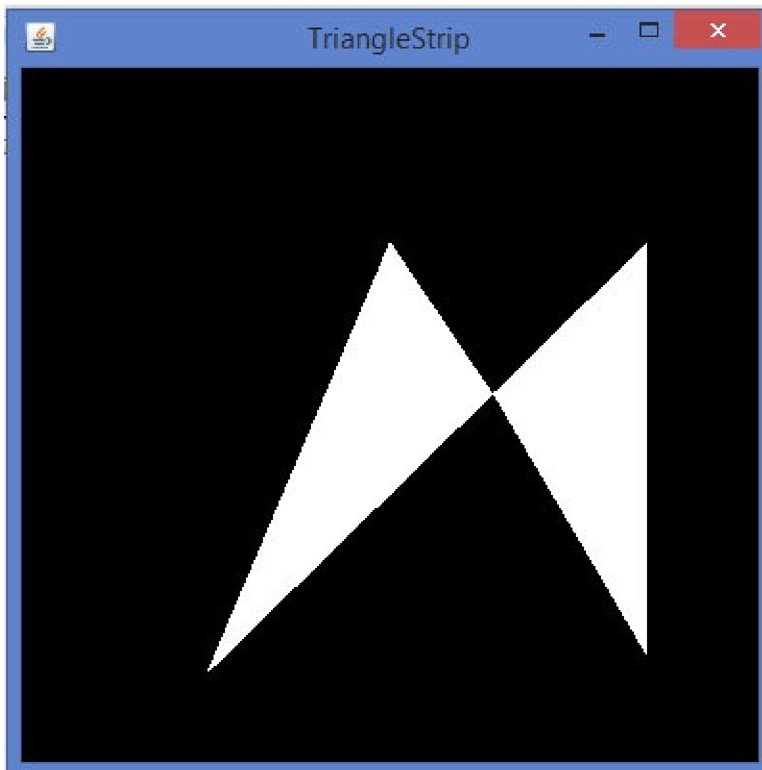
如果用上面的代码替换显示任何基本的模板程序的方法，编译并执行它，下面的输出生成：




代码片段display()方法来绘制三角形：

```
public void display(GLAutoDrawable drawable) {
   final GL2 gl = drawable.getGL().getGL2();
   gl.glBegin (GL2.GL_TRIANGLE_STRIP);
   gl.glVertex3f(0f,0.5f,0);
   gl.glVertex3f(-0.50f,-0.75f,0);
   gl.glVertex3f(0.28f,0.06f,0);
   gl.glVertex3f(0.7f,0.5f,0);
   gl.glVertex3f(0.7f,-0.7f,0);
   gl.glEnd();
}
```

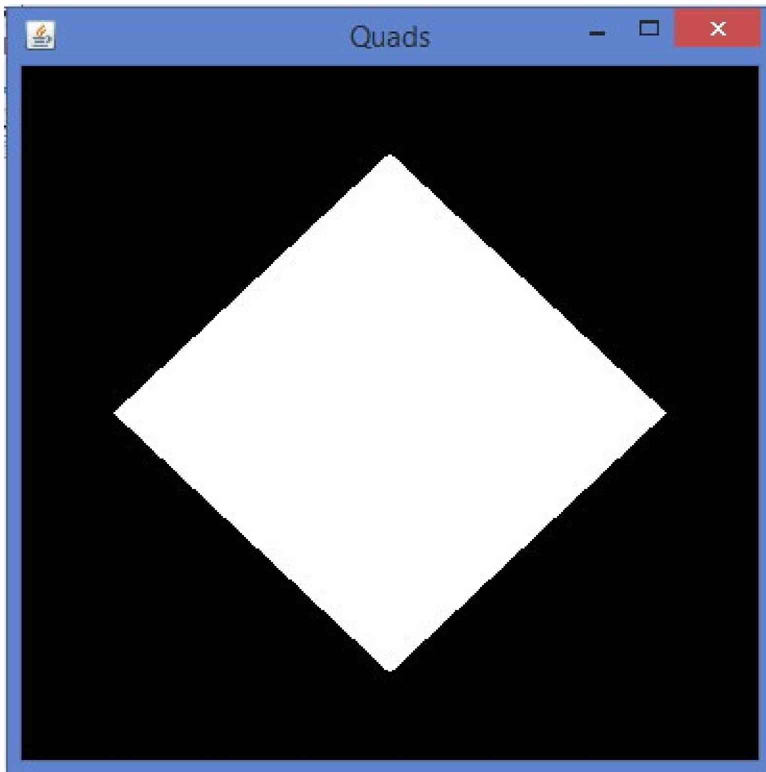
如果要更换显示器的任何与上面的代码的基本模板方案的方法，编译并执行它，下面的输出生成：




代码片段display()方法来绘制四边形：

```
public void display(GLAutoDrawable drawable) {
   final GL2 gl = drawable.getGL().getGL2();
   gl.glBegin(GL2.GL_QUADS);
   gl.glVertex3f( 0.0f,0.75f,0);
   gl.glVertex3f(-0.75f,0f,0);
   gl.glVertex3f(0f,-0.75f,0);
   gl.glVertex3f(0.75f,0f,0);
   gl.glEnd();
}
```

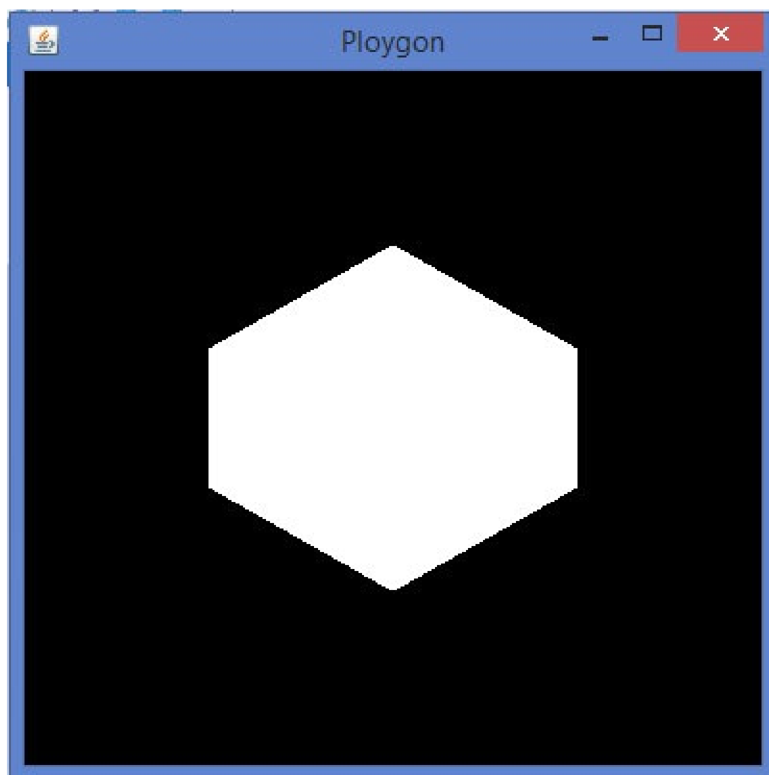
如果用上面的代码替换显示任何基本的模板程序的方法，编译并执行它，下面的输出生成：




代码片段display()方法来绘制多边形：

```
public void display(GLAutoDrawable drawable) {
   final GL2 gl = drawable.getGL().getGL2();
   gl.glBegin(GL2.GL_POLYGON);
   gl.glVertex3f(0f,0.5f,0f);
   gl.glVertex3f(-0.5f,0.2f,0f);
   gl.glVertex3f(-0.5f,-0.2f,0f);
   gl.glVertex3f(0f,-0.5f,0f);
   gl.glVertex3f(0f,0.5f,0f);
   gl.glVertex3f(0.5f,0.2f,0f);
   gl.glVertex3f(0.5f,-0.2f,0f);
   gl.glVertex3f(0f,-0.5f,0f);
   gl.glEnd();
}
```

如果用上面的代码替换任何基本的模板方案的display()方法，编译并执行它，会生成以下输出


Ploygon

# JOGL转化对象 - JOGL教程

OpenGL提供了更多的功能，比如色彩应用到一个对象，比例，灯光，旋转的物体等。本章介绍了一些变换使用JOGL的对象。

## 移动对象的窗口

在前面的章节中，我们讨论的方案画线，用简单的线条绘制各种形状。以这种方式创建的形状可被显示在该窗口内的任何位置。它是通过使用glTranslatef (float x, float y, float z) 方法完成。

这种方法属于GLMatrixFunc接口，它在javax.media.opengl.fixedfunc包。

### GLMatrixFunc 接口

interface: GLMatrixFunc

package: javax.media.opengl.fixedfunc

让我们来看看这个接口的一些重要方法：

| Sr. No. | 方法和说明 |
|---|---|
| 1 | **void glRotatef(float angle, float x, float y, float z)**这个方法旋转当前矩阵。 |
| 2 | **void glScalef(float x, float y, float z)**此方法用于缩放当前矩阵。 |
| 3 | **void glTranslatef(float x, float y,float z)**此方法用于转换的当前矩阵。 |
| 4 | **void glLoadIdentity()**此方法加载当前矩阵与单位矩阵。 |

glTranslate()方法的移动坐标系的原点，以通过所述参数（x，y，z），传递给glTranslate()方法作为参数指定的点。保存和恢复的未翻译的坐标系，glPushMatrix()和glPopMatrix()方法被使用。

```
gl.glTranslatef(0f, 0f, -2.5f);
```

只要 glTranslate()方法被使用时，它改变了组件的屏幕上的位置。因此，GLEventListener 界面的 reshape()方法应该重写和OpenGL视口和投影矩阵应该初始化。

下面的代码显示初始化视口和投影矩阵模板：

```
public void reshape(GLAutoDrawable drawable, int x,  int y, int wic
   final GL2 gl = drawable.getGL().getGL2();
   // get the OpenGL 2 graphics object
   if(height <=0)
      height =1;
   //preventing devided by 0 exception height =1;
   final float h = (float) width / (float) height;
   // display area to cover the entire window
   gl.glViewport(0, 0, width, height);
   //transforming projection matrix
   gl.glMatrixMode(GL2.GL_PROJECTION);
   gl.glLoadIdentity();
   glu.gluPerspective(45.0f, h, 1.0, 20.0);
   //transforming model view gl.glLoadIdentity();
   gl.glMatrixMode(GL2.GL_MODELVIEW);
   gl.glLoadIdentity();
}
```

## 运用颜色的对象

要应用颜色的物体，使用GL2类的glColor()方法。

语法

```
gl.glColorXY(1f,0f,0f);
```

例子

如果通过颜色值（1，0，0），那么得到的红色和（1，1，0）的值给定为黄色。

- x表示使用的颜色数， 3 (red, blue, green) or 4(red, blue, green, alpha)。为了得到不同的颜色组合，这些颜色值作为参数传递。颜色参数的序列必须是维护的顺序。
- y表示它接受的参数，如字节byte(b), double(d), float(f), int(i), short(s), ubyte(ub), uint(ui), ushort(us)。

```
gl.glColor3f(1f,0f,0f);   //gives us red
gl.glColor3f(0f,1f,0f);   //gives us blue
gl.glColor3f(0f,0f,1f);   //gives us green
```

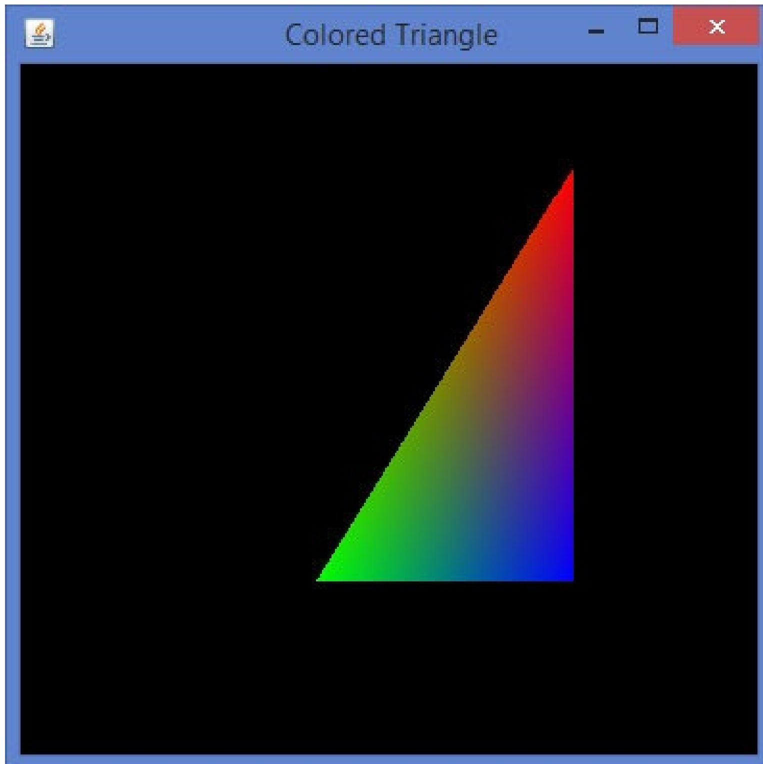
如果三角形，可以为每个顶点应用不同的颜色。

让我们通过程序的颜色应用到一个三角形：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class TriangleColor implements GLEventListener{
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glBegin( GL2.GL_TRIANGLES );        // Drawing Using Tria
      gl.glColor3f( 1.0f, 0.0f, 0.0f ); //Red
      gl.glVertex3f( 0.5f,0.7f,0.0f ); // Top
      gl.glColor3f( 0.0f,1.0f,0.0f ); //blue
      gl.glVertex3f( -0.2f,-0.50f,0.0f );              // Bottom Le
      gl.glColor3f( 0.0f,0.0f,1.0f ); //green
      gl.glVertex3f( 0.5f,-0.5f,0.0f );              //Bottom Right
      gl.glEnd();
   }
   @Override
   public void dispose( GLAutoDrawable arg0 ) {
      //method body
   }
   @Override
   public void init( GLAutoDrawable arg0 ) {
      // method body
   }
   @Override
   public void reshape( GLAutoDrawable arg0, int arg1, int arg2, in
      // method body
   }
   public static void main( String[] args ) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get( GLProfile.GL2 );
      GLCapabilities capabilities = new GLCapabilities( profile );
      // The canvas
      final GLCanvas glcanvas = new GLCanvas( capabilities );
      TriangleColor triangle = new TriangleColor();
      glcanvas.addGLEventListener( triangle );
      glcanvas.setSize( 400, 400 );
      //creating frame
      final JFrame frame = new JFrame (" Colored Triangle");
      //adding canvas to it
      frame.getContentPane().add( glcanvas );
      frame.setSize( frame.getContentPane().getPreferredSize() );
      frame.setVisible( true );
   }//end of main
}//end of class
```

当编译并执行以上程序，会得到如下彩色三角形：




## 应用颜色为多边形

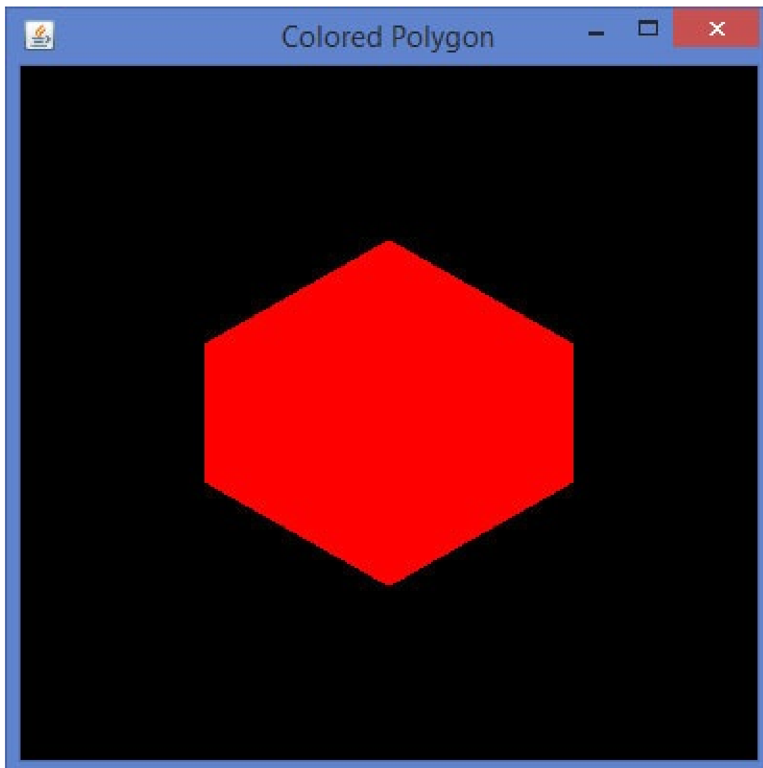
让我们通过程序的颜色应用到多边形：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class PolygonColor implements GLEventListener{
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glColor3f( 1f,0f,0f ); //applying red
      gl.glBegin( GL2.GL_POLYGON );
      gl.glVertex3f( 0f,0.5f,0f  );
      gl.glVertex3f( -0.5f,0.2f,0f );
      gl.glVertex3f( -0.5f,-0.2f,0f );
      gl.glVertex3f( 0f,-0.5f,0f );
      gl.glVertex3f( 0f,0.5f,0f );
      gl.glVertex3f( 0.5f,0.2f,0f );
      gl.glVertex3f( 0.5f,-0.2f,0f );
      gl.glVertex3f( 0f,-0.5f,0f );
      gl.glEnd();
```

```
    }
    @Override
    public void dispose( GLAutoDrawable arg0 ) {
       //method body
    }
    @Override
    public void init( GLAutoDrawable arg0 ) {
       // method body
    }
    @Override
    public void reshape( GLAutoDrawable arg0, int arg1, int arg2, ir
       // method body
    }
    public static void main( String[] args ) {
       //getting the capabilities object of GL2 profile
       final GLProfile profile = GLProfile.get( GLProfile.GL2 );
       GLCapabilities capabilities = new GLCapabilities( profile );
       // The canvas
       final GLCanvas glcanvas = new GLCanvas( capabilities );
       PolygonColor polygon = new PolygonColor();
       glcanvas.addGLEventListener( polygon );
       glcanvas.setSize( 400, 400 );
       //creating frame
       final JFrame frame = new JFrame ( "Colored Polygon" );
       //adding canvas to frame
       frame.getContentPane().add( glcanvas );
       frame.setSize( frame.getContentPane().getPreferredSize() );
       frame.setVisible( true );
    }//end of main
 }//end of class
```

当编译并执行上述程序，将生成以下输出：




# 缩放

缩放对象是通过使用GLMatrixFunc接口的void glScalef(float x, float y, float z) 方法进行。该方法接受三个浮点参数，使用我们指定轴沿x，y和z比例因子。

例如，在下面的程序中，一个三角形是减弱至50%。在这里，50传递的是沿所有轴的参数。

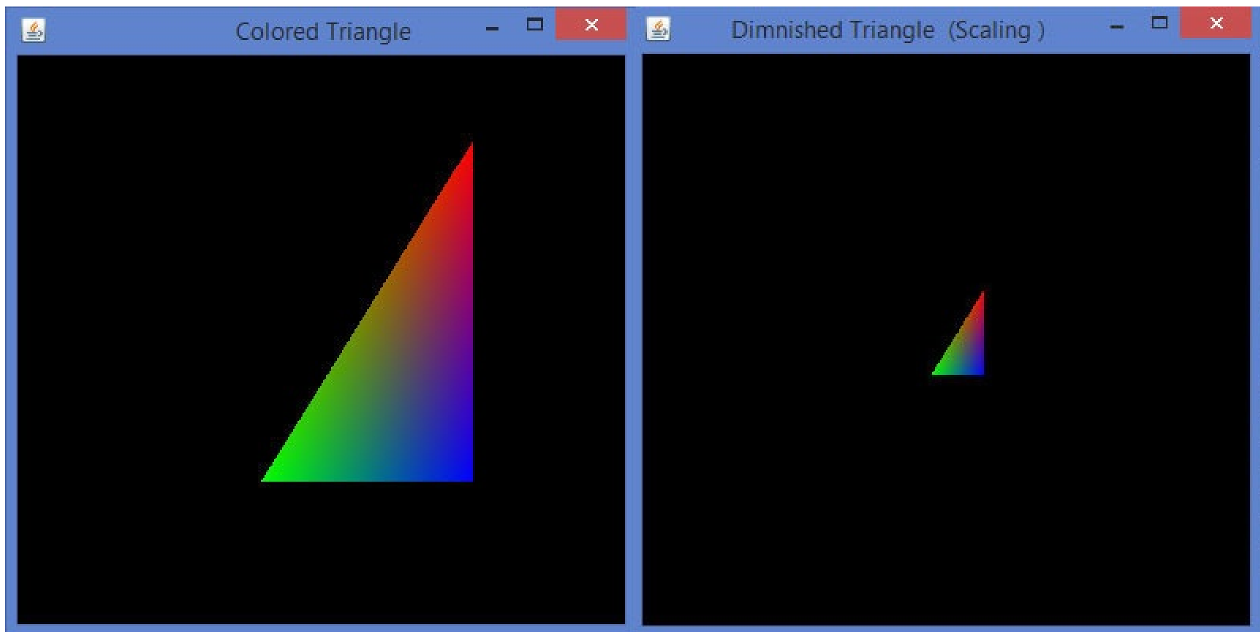
让我们通过程序来扩展一个三角形：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;

public class Scaling implements GLEventListener{
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glScalef( 0.50f,0.25f,0.50f );
      gl.glBegin( GL2.GL_TRIANGLES );       // Drawing Using Triang
      gl.glColor3f( 1.0f, 0.0f, 0.0f ); //Red
      gl.glVertex3f( 0.5f,0.7f,0.0f ); // Top
      gl.glColor3f( 0.0f,1.0f,0.0f ); //blue
      gl.glVertex3f( -0.2f,-0.50f,0.0f );          // Bottom Left
      gl.glColor3f( 0.0f,0.0f,1.0f ); //green
```

```
      gl.glVertex3f( 0.5f,-0.5f,0.0f );           //Bottom Right
      gl.glEnd();
   }
   @Override
   public void dispose( GLAutoDrawable arg0 ) {
      //method body
   }
   @Override
   public void init( GLAutoDrawable arg0 ) {
      // method body
   }
   @Override
   public void reshape( GLAutoDrawable arg0, int arg1, int arg2, in
      // method body
   }
   public static void main( String[] args ) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get( GLProfile.GL2 );
      GLCapabilities capabilities = new GLCapabilities( profile );
      // The canvas
      final GLCanvas glcanvas = new GLCanvas( capabilities );
      Scaling scaling = new Scaling();
      glcanvas.addGLEventListener( scaling );
      glcanvas.setSize( 400, 400 );
      //creating frame
      final JFrame frame = new JFrame (" Dimnished Triangle (Scalin
      //adding canvas to it
      frame.getContentPane().add(glcanvas);
      frame.setSize(frame.getContentPane().getPreferredSize());
      frame.setVisible(true);
   }//end of main
}//end of classimport javax.media.opengl.GL2;
```

编译和执行上面的程序，我们得到以下输出。在这里，可以看到一个三角形的减弱相比，由TriangleColor.java生产的原三角形：




# 旋转

对象旋转可以沿任意3轴来完成，使用GLMatrixFunc接口的void glRotatef(float angle, float x, float y, float z) 方法。需要传递的旋转以及x，y，z轴的角度作为参数传递给该方法。

下面的步骤指导成功地旋转对象：

- 清除颜色缓存和深度缓存最初使用gl.glClear（GL2.GL_COLOR_BUFFER_BIT|GL2.GL_DEPTH_BUFFER_BIT）方法。此方法擦除对象的先前状态，使视图清晰。
- 复位用glLoadIdentity()方法的投影矩阵。

实例化的动画类和使用start()方法启动动画。

## FPSAnimator 类

Class:

FPSAnimator

Package: javax.media.opengl.util

构造方法

```
FPSAnimator(GLAutoDrawable drawable, int fps)
```

创建给定的目标帧每秒的值和初始绘制的动画一个FPSAnimator。

```
FPSAnimator(GLAutoDrawable drawable, int fps, boolean cheduleAtFixe
```

创建一个具有给定的目标帧每秒的值，初始绘制动画，和一个标志，指示是否使用固定利率调度FPSAnimator。

```
FPSAnimator(int fps)
```

创建由给定的目标帧每秒值的FPSAnimator。

```
FPSAnimator(int fps, boolean scheduleAtFixedRate)
```

创建由给定的目标帧每秒的值和一个标志，指示是否使用固定速率的调度FPSAnimator。

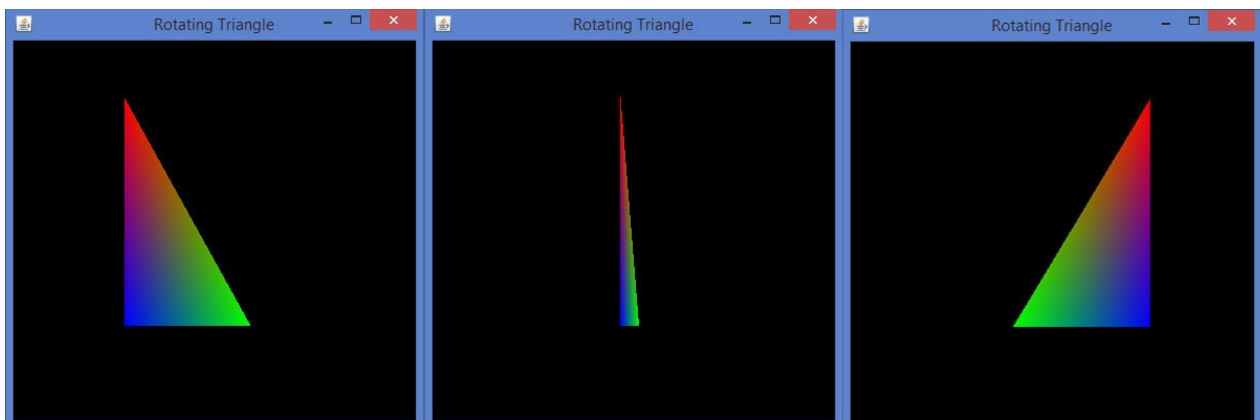
start() 和stop()在这个类中的两个重要的方法。

让我们通过程序来旋转一个三角形：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;
import com.jogamp.opengl.util.FPSAnimator;
public class TriangleRotation implements GLEventListener{
   private float rtri;  //for angle of rotation
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glClear (GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BI
      // Clear The Screen And The Depth Buffer
      gl.glLoadIdentity();                  // Reset The View
      gl.glRotatef( rtri, 0.0f, 1.0f, 0.0f );//triangle rotation
      gl.glBegin( GL2.GL_TRIANGLES );         // Drawing Using Tria
      gl.glColor3f( 1.0f, 0.0f, 0.0f ); //Red
      gl.glVertex3f( 0.5f,0.7f,0.0f ); // Top
      gl.glColor3f( 0.0f,1.0f,0.0f ); //blue
      gl.glVertex3f( -0.2f,-0.50f,0.0f );              // Bottom Le
      gl.glColor3f( 0.0f,0.0f,1.0f ); //green
      gl.glVertex3f( 0.5f,-0.5f,0.0f );              // Bottom Righ
      gl.glEnd();
      gl.glFlush();
      rtri +=0.2f;  //assigning the angle
   }
   @Override
```

```
    public void dispose( GLAutoDrawable arg0 ) {
       //method body
    }
    @Override
    public void init( GLAutoDrawable arg0 ) {
       // method body
    }
    @Override
    public void reshape( GLAutoDrawable drawable, int x, int y, int
       // method body
    }
    public static void main( String[] args ) {
       //getting the capabilities object of GL2 profile
       final GLProfile profile = GLProfile.get( GLProfile.GL2 );
       GLCapabilities capabilities = new GLCapabilities( profile );
       // The canvas
       final GLCanvas glcanvas = new GLCanvas( capabilities );
       TriangleRotation triangle = new TriangleRotation();
       glcanvas.addGLEventListener( triangle );
       glcanvas.setSize( 400, 400 );
       //creating frame
       final JFrame frame = new JFrame ( "Rotating Triangle" );
       //adding canvas to it
       frame.getContentPane().add( glcanvas );
       frame.setSize( frame.getContentPane().getPreferredSize() );
       frame.setVisible( true );
       //Instantiating and Initiating Animator
       final FPSAnimator animator = new FPSAnimator( glcanvas, 300,t
       animator.start();
    }//end of main
 }//end of class
```

如果编译并执行上述程序，它会生成以下输出。在这里，可以看到周围x轴旋转彩色三角形的各个快照。




# 灯光

要设置灯光，使用过glEnable()方法初步启用的照明。然后应用照明的对象，使用GLLightingFunc接口的 glLightfv(int light, int pname, float[] params, int params_offset) 方法。这个方法有四个参数。

下表描述了gllightfv()方法的参数。

| Sr. No. | 参数名称和描述 |
|---|---|
| 1 | **Light**指定的光。灯的数量依赖于实现，但至少八个灯支持。它接受10个值，这些参数是在一个名为下面给出的光源参数表中单独讨论。 |
| 2 | **Pname**指定一个单值的光源参数。光源有10个参数，如下所述。 |
| 3 | **Params**指定的指针被设置到的光源的参数pname的一个或多个值。 |
| 4 | **Light source parameter**可以使用以下任何给定的光源参数。 |

光源参数：

| Sr. No. | 参数及描述 |
|---|---|
| 1 | **GL_AMBIENT**它包含指定的光的环境亮度的参数。 |
| 2 | **GL_DIFFUSE**它包含指定的光的漫反射的强度的参数。 |
| 3 | **GL_SPECULAR**它包含指定的光的镜面反射强度的参数。 |
| 4 | **GL_POSITION**它包含指定的均质物体坐标的光的位置的4个整数或浮点值。 |
| 5 | **GL_SPOT_DIRECTION**它包含在均质物体坐标指定的光的方向的参数。 |
| 6 | **GL_SPOT_EXPONENT**此参数指定的光的强度分布。 |
| 7 | **GL_SPOT_CUTOFF**这种单参数指定的光的最大发散角。 |
| 8 | **GL_CONSTANT_ATTENUATION or GL_LINEAR_ATTENUATION or GL_QUADRATIC_ATTENUATION**可以使用任意的衰减系数，它是由一个单一的值来表示。 |

照明被启用并使用过glEnable()方法和glDisable()函数以及参数GL_LIGHTING禁用。

下面的模板，给出了灯光：

```
gl.glEnable(GL2.GL_LIGHTING);
gl.glEnable(GL2.GL_LIGHT0); gl.glEnable(GL2.GL_NORMALIZE);
float[] ambientLight = { 0.1f, 0.f, 0.f,0f }; // weak RED ambient
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_AMBIENT, ambientLight, 0);
float[] diffuseLight = { 1f,2f,1f,0f }; //multi-color diffuse
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_DIFFUSE, diffuseLight, 0);
```

# 施加光到一个旋转多角

遵循用于将光以一个旋转多角给定的步骤。

使用旋转glRotate()方法的多边形：

```
gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);
// Clear The Screen And The Depth Buffer
gl.glLoadIdentity();
// Reset The View
gl.glRotatef(rpoly, 0.0f, 1.0f, 0.0f);
```

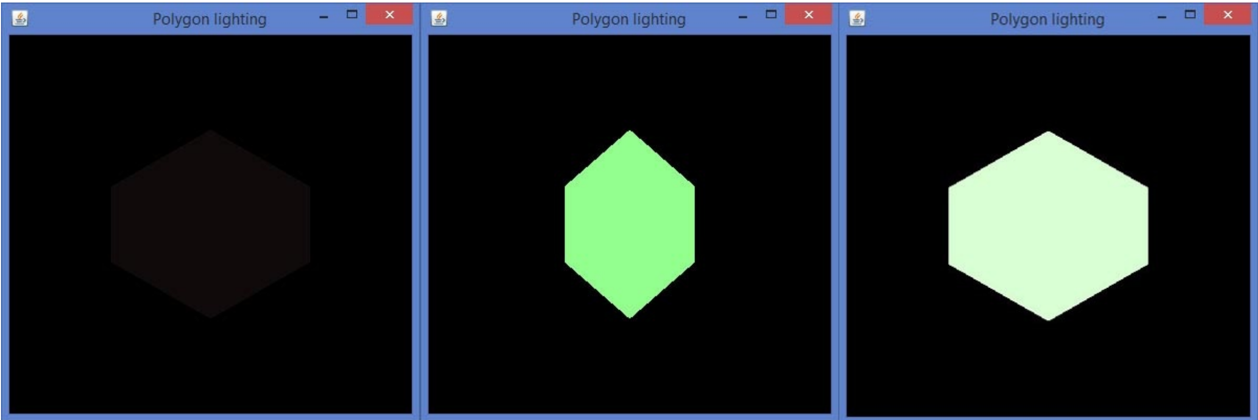
让我们通过程序将光应用到旋转面：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.swing.JFrame;
import com.jogamp.opengl.util.FPSAnimator;

public class PolygonLighting implements GLEventListener{
   private float rpoly;
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glColor3f(1f,0f,0f); //applying red
      // Clear The Screen And The Depth Buffer
      gl.glClear( GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT
      gl.glLoadIdentity();                  // Reset The View
      gl.glRotatef( rpoly, 0.0f, 1.0f, 0.0f );
      gl.glBegin( GL2.GL_POLYGON );
      gl.glVertex3f( 0f,0.5f,0f );
      gl.glVertex3f( -0.5f,0.2f,0f );
      gl.glVertex3f( -0.5f,-0.2f,0f );
      gl.glVertex3f( 0f,-0.5f,0f );
      gl.glVertex3f( 0f,0.5f,0f );
      gl.glVertex3f( 0.5f,0.2f,0f );
```

```
      gl.glVertex3f( 0.5f,-0.2f,0f );
      gl.glVertex3f( 0f,-0.5f,0f );
      gl.glEnd();
      gl.glFlush();
      rpoly +=0.2f;  //assigning the angle
      gl.glEnable( GL2.GL_LIGHTING );
      gl.glEnable( GL2.GL_LIGHT0 );
      gl.glEnable( GL2.GL_NORMALIZE );
      float[] ambientLight =  0.1f, 0.f, 0.f,0f }; // weak RED amb
      gl.glLightfv( GL2.GL_LIGHT0, GL2.GL_AMBIENT, ambient-Light, (
      float[] diffuseLight = { 1f,2f,1f,0f }; //multi color diffuse
      gl.glLightfv( GL2.GL_LIGHT0, GL2.GL_DIFFUSE, diffuse-Light, (
   }
   @Override
   public void dispose( GLAutoDrawable arg0 ) {
      //method body
   }
   @Override
   public void init( GLAutoDrawable arg0 ) {
      // method body
   }
   @Override
   public void reshape( GLAutoDrawable arg0, int arg1, int arg2, in
      // method body
   }
   public static void main( String[] args ) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get( GLProfile.GL2 );
      GLCapabilities capabilities = new GLCapabilities( profile );
      // The canvas
      final GLCanvas glcanvas = new GLCanvas( capabilities );
      PolygonLighting polygonlighting = new PolygonLighting();
      glcanvas.addGLEventListener( polygonlighting );
      glcanvas.setSize( 400, 400 );
      //creating frame
      final JFrame frame = new JFrame ( " Polygon lighting " );
      //adding canvas to it
      frame.getContentPane().add( glcanvas );
      frame.setSize( frame.getContentPane().getPreferredSize() );
      frame.setVisible( true );
      //Instantiating and Initiating Animator
      final FPSAnimator animator = new FPSAnimator(glcanvas, 300,tr
      animator.start();
   }//end of main
}//end of class
```

如果编译并执行上述程序，它会生成以下输出。在这里，可以观察到一个旋转的多边形灯光的各种快照。


Polygon lighting
Polygon lighting
Polygon lighting

# JOGL 3D图形 - JOGL教程

在本章中，让我们来学习如何处理3D图形。

## 绘制3D线

让我们绘制与z轴成简单的线，看到2D和3D线之间的差值。先画一条简单的直线，再画第二条线3个单元到窗口中。

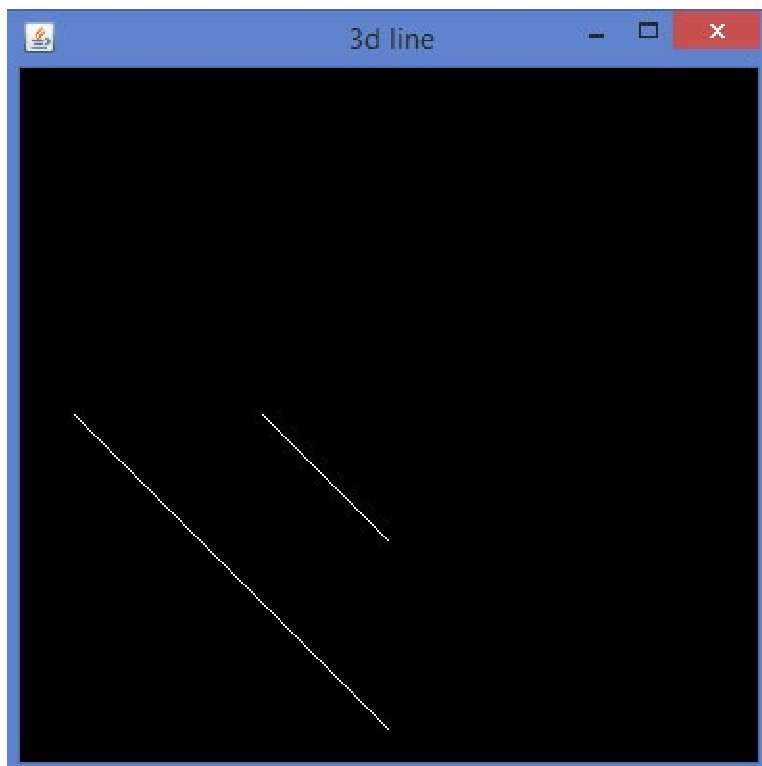
让我们通过程序来绘制3D线：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.glu.GLU;
import javax.swing.JFrame;

public class Line3d implements GLEventListener{
   private GLU glu = new GLU();
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glTranslatef( 0f, 0f, -2.5f );
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( -0.75f,0f,0 );
      gl.glVertex3f( 0f,-0.75f, 0 );
      gl.glEnd();
      //3d line
      gl.glBegin( GL2.GL_LINES );
      gl.glVertex3f( -0.75f,0f,3f );// 3 units into the window
      gl.glVertex3f( 0f,-0.75f,3f );
      gl.glEnd();
   }
   @Override
   public void dispose( GLAutoDrawable arg0 ) {
      //method body
   }
   @Override
   public void init( GLAutoDrawable arg0 ) {
      // method body
   }
   @Override
   public void reshape( GLAutoDrawable drawable, int x, int y, int
      GL2 gl = drawable.getGL().getGL2();
      if( height <=0 )
```

```
         height =1;
      final float h = ( float ) width / ( float ) height;
      gl.glViewport( 0, 0, width, height );
      gl.glMatrixMode( GL2.GL_PROJECTION );
      gl.glLoadIdentity();
      glu.gluPerspective( 45.0f, h, 1.0, 20.0 );
      gl.glMatrixMode( GL2.GL_MODELVIEW );
      gl.glLoadIdentity();
   }
   public static void main( String[] args ) {
      //getting the capabilities object of GL2 profile
      final GLProfile profile = GLProfile.get( GLProfile.GL2 );
      GLCapabilities capabilities = new GLCapabilities( profile );
      // The canvas
      final GLCanvas glcanvas = new GLCanvas( capabilities );
      Line3d line3d = new Line3d();
      glcanvas.addGLEventListener( line3d );
      glcanvas.setSize( 400, 400 );
      //creating frame
      final JFrame frame = new JFrame (" 3d line");
      //adding canvas to it
      frame.getContentPane().add( glcanvas );
      frame.setSize( frame.getContentPane().getPreferredSize() );
      frame.setVisible( true );
   }//end of main
}//end of class
```

当编译并执行上述程序，将生成以下输出：



3D形状可以通过glVertex3f()方法，该方法产生上述观点的Z-象限赋予非零值绘制。现在加入剩余行会导致一个三维边缘。

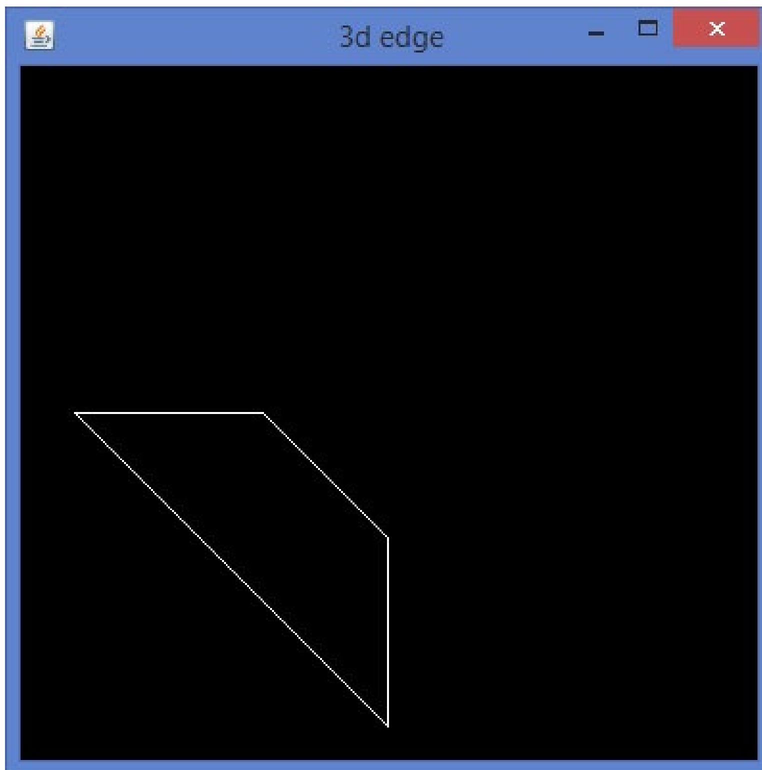
程序开发3D优势：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.glu.GLU;
import javax.swing.JFrame;

public class Edge1 implements GLEventListener{
   private GLU glu = new GLU();
   @Override
   public void display(GLAutoDrawable drawable) {
      // TODO Auto-generated method stub
      final GL2 gl = drawable.getGL().getGL2();
      gl.glTranslatef(0f, 0f, -2.5f);
      gl.glBegin(GL2.GL_LINES);
      gl.glVertex3f(-0.75f,0f,0);
      gl.glVertex3f(0f,-0.75f, 0);
      gl.glEnd();
      //3d line
      gl.glBegin(GL2.GL_LINES);
      //3 units in to the window
      gl.glVertex3f(-0.75f,0f,3f);
      gl.glVertex3f(0f,-0.75f,3f);
      gl.glEnd();
      //top
      gl.glBegin(GL2.GL_LINES);
      gl.glVertex3f(-0.75f,0f,0);
      gl.glVertex3f(-0.75f,0f,3f);
      gl.glEnd();
      //bottom
      gl.glBegin(GL2.GL_LINES);
      gl.glVertex3f(0f,-0.75f, 0);
      gl.glVertex3f(0f,-0.75f,3f);
      gl.glEnd();
   }
   @Override
   public void dispose(GLAutoDrawable arg0) {
      //method body
   }
   @Override
   public void init(GLAutoDrawable arg0) {
      // method body
   }
   @Override
   public void reshape(GLAutoDrawable drawable, int x, int y, int w
```

```
        GL2 gl = drawable.getGL().getGL2();
        if(height <=0)
           height =1;
        final float h = (float) width / (float) height;
        gl.glViewport(0, 0, width, height);
        gl.glMatrixMode(GL2.GL_PROJECTION);
        gl.glLoadIdentity();
        glu.gluPerspective(45.0f, h, 1.0, 20.0);
        gl.glMatrixMode(GL2.GL_MODELVIEW);
        gl.glLoadIdentity();
     }
     public static void main(String[] args) {
        //getting the capabilities object of GL2 profile
        final GLProfile profile = GLProfile.get(GLProfile.GL2);
        GLCapabilities capabilities = new GLCapabilities(profile);
        // The canvas
        final GLCanvas glcanvas = new GLCanvas(capabilities);
        Edge1 b = new Edge1();
        glcanvas.addGLEventListener(b);
        glcanvas.setSize(400, 400);
        //creating frame
        final JFrame frame = new JFrame (" 3d edge");
        //adding canvas to it
        frame.getContentPane().add(glcanvas);
        frame.setSize(frame.getContentPane().getPreferredSize());
        frame.setVisible(true);
     }//end of main
  }//end of class
```

当编译并执行上述程序，将生成以下输出：




以同样的方式，由显影3D边缘到对应的任何二维四边形的侧面和连接相邻顶点，就可以得到一个3D四边形。

让我们通过程序来绘制一个菱形：

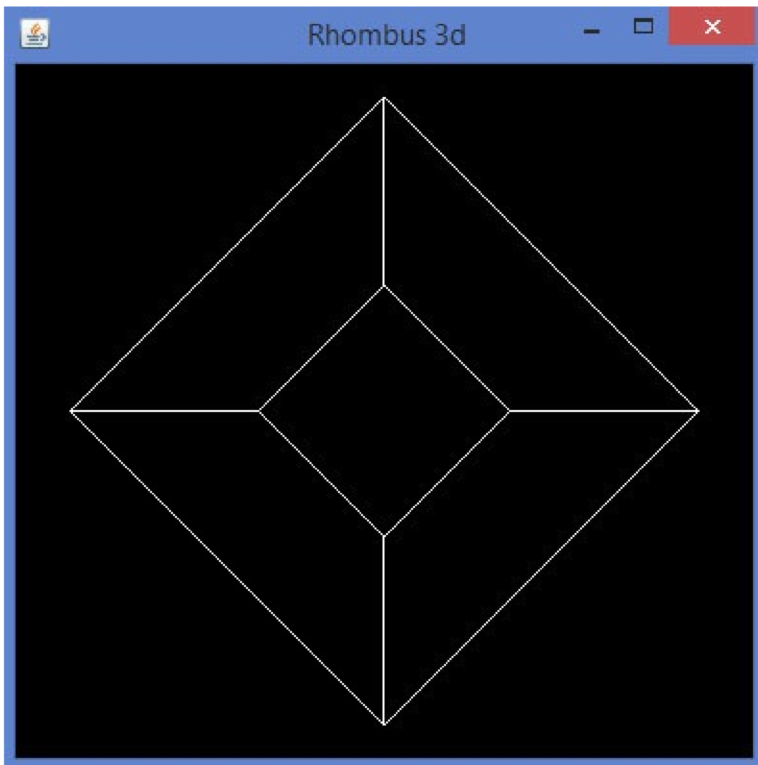
```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.glu.GLU;
import javax.swing.JFrame;

public class Rhombus implements GLEventListener{
   private GLU glu = new GLU();
   @Override
   public void display(GLAutoDrawable drawable) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glTranslatef(0f, 0f, -2.5f);
      //drawing edge1.....
      gl.glBegin(GL2.GL_LINES);
      gl.glVertex3f(-0.75f,0f,0);
      gl.glVertex3f(0f,-0.75f, 0);
      gl.glEnd();
      gl.glBegin(GL2.GL_LINES);
      gl.glVertex3f(-0.75f,0f,3f);// 3 units into the window
      gl.glVertex3f(0f,-0.75f,3f);
      gl.glEnd();
      //top
```

```
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(-0.75f,0f,0);
        gl.glVertex3f(-0.75f,0f,3f);
        gl.glEnd();
        //bottom
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(0f,-0.75f, 0);
        gl.glVertex3f(0f,-0.75f,3f);
        gl.glEnd();
        //edge 2....
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(0f,-0.75f, 0);
        gl.glVertex3f(0.75f,0f, 0);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(0f,-0.75f, 3f);
        gl.glVertex3f(0.75f,0f, 3f);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(0f,-0.75f, 0);
        gl.glVertex3f(0f,-0.75f, 3f);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(0.75f,0f, 0);
        gl.glVertex3f(0.75f,0f, 3f);
        gl.glEnd();
        //Edge 3.............
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f( 0.0f,0.75f,0);
        gl.glVertex3f(-0.75f,0f,0);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f( 0.0f,0.75f,3f);
        gl.glVertex3f(-0.75f,0f,3f);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f( 0.0f,0.75f,0);
        gl.glVertex3f( 0.0f,0.75f,3f);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(-0.75f,0f,0);
        gl.glVertex3f(-0.75f,0f,3f);
        gl.glEnd();
        //final edge
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(0.75f,0f, 0);
        gl.glVertex3f( 0.0f,0.75f,0);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f(0.75f,0f,3f);
        gl.glVertex3f( 0.0f,0.75f,3f);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
```

```
        gl.glVertex3f(0.75f,0f, 0);
        gl.glVertex3f(0.75f,0f,3f);
        gl.glEnd();
        gl.glBegin(GL2.GL_LINES);
        gl.glVertex3f( 0.0f,0.75f,0);
        gl.glVertex3f( 0.0f,0.75f,3f);
        gl.glEnd();
     }
    @Override
    public void dispose(GLAutoDrawable arg0) {
       //method body
    }
    @Override
    public void init(GLAutoDrawable arg0) {
       // method body
    }
    @Override
    public void reshape(GLAutoDrawable drawable, int x, int y, int w
       // TODO Auto-generated method stub final
       GL2 gl = drawable.getGL().getGL2();
       if(height <=0)
       height =1;
       final float h = (float) width / (float) height;
       gl.glViewport(3, 6, width, height);
       gl.glMatrixMode(GL2.GL_PROJECTION);
       gl.glLoadIdentity();
       glu.gluPerspective(45.0f, h, 1.0, 20.0);
       gl.glMatrixMode(GL2.GL_MODELVIEW);
       gl.glLoadIdentity();
    }
    public static void main(String[] args) {
       //getting the capabilities object of GL2 profile
       final GLProfile profile = GLProfile.get(GLProfile.GL2);
       GLCapabilities capabilities = new GLCapabilities(profile);
       // The canvas
       final GLCanvas glcanvas = new GLCanvas(capabilities);
       Rhombus b = new Rhombus();
       glcanvas.addGLEventListener(b);
       glcanvas.setSize(400, 400);
       //creating frame
       final JFrame frame = new JFrame (" Rhombus 3d");
       //adding canvas to it
       frame.getContentPane().add(glcanvas);
       frame.setSize(frame.getContentPane().getPreferredSize());
       frame.setVisible(true);
    }//end of main
 }//end of classimport javax.media.opengl.GL2;
```

当编译并执行上述程序，将生成以下输出。它显示了一个使用3D线条绘制的菱形




glBegin()方法的预定义参数可用于绘制3D形状。

让我们通过程序绘制一个3D三角（无深度测试）：

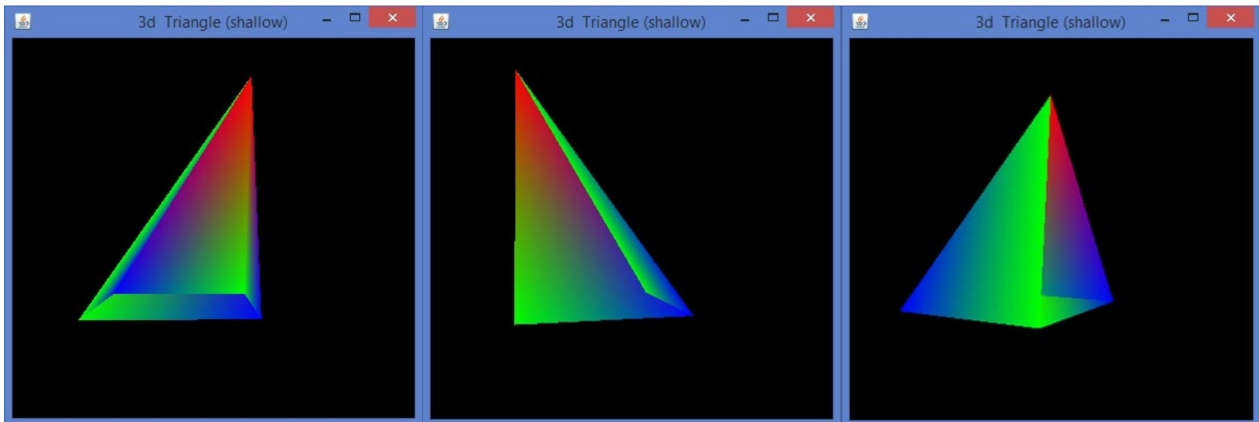
```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.glu.GLU;
import javax.swing.JFrame;
import com.jogamp.opengl.util.FPSAnimator;

public class Triangle3d implements GLEventListener{
   private GLU glu = new GLU();
   private float rtri =0.0f;
   @Override
   public void display(GLAutoDrawable drawable) {
      final GL2 gl = drawable.getGL().getGL2();
      // Clear The Screen And The Depth Buffer
      gl.glClear( GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BI
      gl.glLoadIdentity();                              // Reset The View
      gl.glTranslatef( -0.5f, 0.0f, -6.0f );           // Move the tr
      gl.glRotatef( rtri, 0.0f, 1.0f, 0.0f );
      gl.glBegin(  GL2.GL_TRIANGLES );
      //drawing triangle in all dimensions
      // Front
      gl.glColor3f( 1.0f, 0.0f, 0.0f ); // Red
      gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top Of Triangle (Front)
      gl.glColor3f( 0.0f, 1.0f, 0.0f ); // Green
```

```
        gl.glVertex3f( -1.0f, -1.0f, 1.0f ); // Left Of Triangle (Fro
        gl.glColor3f( 0.0f, 0.0f, 1.0f ); // Blue
        gl.glVertex3f( 1.0f, -1.0f, 1.0f ); // Right Of Triangle (Fro
        // Right
        gl.glColor3f( 1.0f, 0.0f, 0.0f ); // Red
        gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top Of Triangle (Right)
        gl.glColor3f( 0.0f, 0.0f, 1.0f ); // Blue
        gl.glVertex3f( 1.0f, -1.0f, 1.0f ); // Left Of Triangle (Righ
        gl.glColor3f( 0.0f, 1.0f, 0.0f ); // Green
        gl.glVertex3f( 1.0f, -1.0f, -1.0f ); // Right Of Triangle (Ri
        // Left
        gl.glColor3f( 1.0f, 0.0f, 0.0f ); // Red
        gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top Of Triangle (Back)
        gl.glColor3f( 0.0f, 1.0f, 0.0f ); // Green
        gl.glVertex3f( 1.0f, -1.0f, -1.0f ); // Left Of Triangle (Bac
        gl.glColor3f( 0.0f, 0.0f, 1.0f ); // Blue
        gl.glVertex3f( -1.0f, -1.0f, -1.0f ); // Right Of Triangle (B
        //left
        gl.glColor3f( 0.0f, 1.0f, 0.0f ); // Red
        gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top Of Triangle (Left)
        gl.glColor3f( 0.0f, 0.0f, 1.0f ); // Blue
        gl.glVertex3f( -1.0f, -1.0f, -1.0f ); // Left Of Triangle (Le
        gl.glColor3f( 0.0f, 1.0f, 0.0f ); // Green
        gl.glVertex3f( -1.0f, -1.0f, 1.0f ); // Right Of Triangle (Le
        gl.glEnd(); // Done Drawing 3d  triangle (Pyramid)
        gl.glFlush();
        rtri +=0.2f;
     }
     @Override
     public void dispose( GLAutoDrawable drawable ) {
        //method body
     }
     @Override
     public void init( GLAutoDrawable drawable ) {
        //method body
     }
     @Override
     public void reshape( GLAutoDrawable drawable, int x, int y, int
        final GL2 gl = drawable.getGL().getGL2();
        if(height <=;)
           height =1;
        final float h = ( float ) width / ( float ) height;
        gl.glViewport( 0, 0, width, height );
        gl.glMatrixMode( GL2.GL_PROJECTION );
        gl.glLoadIdentity();
        glu.gluPerspective( 45.0f, h, 1.0, 20.0 );
        gl.glMatrixMode( GL2.GL_MODELVIEW );
        gl.glLoadIdentity();
     }
     public static void main( String[] args ) {
        final GLProfile profile = GLProfile.get( GLProfile.GL2 );
        GLCapabilities capabilities = new GLCapabilities( profile );
        // The canvas
```

```
        final GLCanvas glcanvas = new GLCanvas( capabilities );
        Triangle3d triangle = new Triangle3d();
        glcanvas.addGLEventListener( triangle );
        glcanvas.setSize( 400, 400 );
        final JFrame frame = new JFrame ( "3d  Triangle (shallow)" );
        frame.getContentPane().add( glcanvas );
        frame.setSize( frame.getContentPane().getPreferredSize() );
        frame.setVisible( true );
        final FPSAnimator animator = new FPSAnimator( glcanvas, 300,t
        animator.start();
    }
}
```

当编译并执行上述程序，将生成以下输出。在这里，有旋转的3D三角形的快照。由于该程序不包含深度测试，三角形生成空洞。




为了使实心的三角形，需要使用过glEnable（GL_DEPTH_TEST），以实现深度测试。启用深度缓冲给出黑屏。方法|这可以同时清除使用glClear（GL_DEPTH_BUFFER_BIT GL_COLOR_BUFFERBIT）的颜色被清除。要启用深度测试中的init()方法或glDisplay()方法，编写如下代码：

```
public void init(GLAutoDrawable drawable) {
   final GL2 gl = drawable.getGL().getGL2();
   gl.glShadeModel(GL2.GL_SMOOTH);
   gl.glClearColor(0f, 0f, 0f, 0f);
   gl.glClearDepth(1.0f);
   gl.glEnable(GL2.GL_DEPTH_TEST);
   gl.glDepthFunc(GL2.GL_LEQUAL);
   gl.glHint(GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL2.GL_NICEST);
}
```

让我们通过程序绘制一个3D三角（深度测试）：

```
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
```

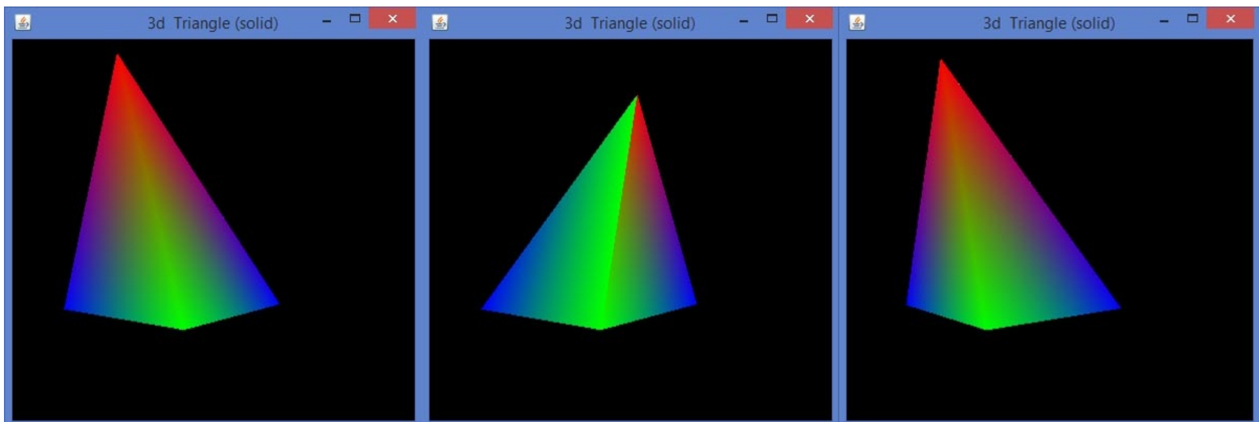
```
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.glu.GLU;
import javax.swing.JFrame;
import com.jogamp.opengl.util.FPSAnimator;

public class Triangledepthtest implements GLEventListener{
   private GLU glu = new GLU();
   private float rtri =0.0f;
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glShadeModel( GL2.GL_SMOOTH );
      gl.glClearColor( 0f, 0f, 0f, 0f );
      gl.glClearDepth( 1.0f );
      gl.glEnable( GL2.GL_DEPTH_TEST );
      gl.glDepthFunc( GL2.GL_LEQUAL );
      gl.glHint( GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL2.GL_NICEST
      // Clear The Screen And The Depth Buffer
      gl.glClear( GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BI
      gl.glLoadIdentity();  // Reset The View
      gl.glTranslatef( -0.5f,0.0f,-6.0f );  // Move the triangle
      gl.glRotatef( rtri, 0.0f, 1.0f, 0.0f );
      gl.glBegin( GL2.GL_TRIANGLES );
      //drawing triangle in all dimentions
      //front
      gl.glColor3f( 1.0f, 0.0f, 0.0f ); // Red
      gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top
      gl.glColor3f( 0.0f, 1.0f, 0.0f ); // Green
      gl.glVertex3f( -1.0f, -1.0f, 1.0f ); // Left
      gl.glColor3f( 0.0f, 0.0f, 1.0f ); // Blue
      gl.glVertex3f( 1.0f, -1.0f, 1.0f ); // Right)
      //right
      gl.glColor3f( 1.0f, 0.0f, 0.0f );
      gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top
      gl.glColor3f( 0.0f, 0.0f, 1.0f );
      gl.glVertex3f( 1.0f, -1.0f, 1.0f ); // Left
      gl.glColor3f( 0.0f, 1.0f, 0.0f );
      gl.glVertex3f( 1.0f, -1.0f, -1.0f ); // Right
      //left
      gl.glColor3f( 1.0f, 0.0f, 0.0f );
      gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top
      gl.glColor3f( 0.0f, 1.0f, 0.0f );
      gl.glVertex3f( 1.0f, -1.0f, -1.0f ); // Left
      gl.glColor3f( 0.0f, 0.0f, 1.0f );
      gl.glVertex3f( -1.0f, -1.0f, -1.0f ); // Right
      //top
      gl.glColor3f( 0.0f, 1.0f, 0.0f );
      gl.glVertex3f( 1.0f, 2.0f, 0.0f ); // Top
      gl.glColor3f( 0.0f, 0.0f, 1.0f );
      gl.glVertex3f( -1.0f, -1.0f, -1.0f ); // Left
      gl.glColor3f( 0.0f, 1.0f, 0.0f );
```

```
         gl.glVertex3f( -1.0f, -1.0f, 1.0f ); // Right
         gl.glEnd(); // Done Drawing 3d  triangle (Pyramid)
         gl.glFlush();
         rtri +=0.2f;
      }
      @Override
      public void dispose( GLAutoDrawable drawable ) {
         //method body
      }
      @Override
      public void init( GLAutoDrawable drawable  ) {
         final GL2 gl = drawable.getGL().getGL2();
         gl.glShadeModel( GL2.GL_SMOOTH );
         gl.glClearColor( 0f, 0f, 0f, 0f );
         gl.glClearDepth( 1.0f );
         gl.glEnable( GL2.GL_DEPTH_TEST );
         gl.glDepthFunc( GL2.GL_LEQUAL );
         gl.glHint( GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL2.GL_NICEST
      }
      @Override
      public void reshape( GLAutoDrawable drawable, int x, int y, int
         final GL2 gl = drawable.getGL().getGL2();
         if( height <=0 )
            height =1;
         final float h = ( float ) width / ( float ) height;
         gl.glViewport( 0, 0, width, height );
         gl.glMatrixMode( GL2.GL_PROJECTION );
         gl.glLoadIdentity();
         glu.gluPerspective( 45.0f, h, 1.0, 20.0 );
         gl.glMatrixMode( GL2.GL_MODELVIEW );
         gl.glLoadIdentity();
      }
      public static void main( String[] args ) {
         // TODO Auto-generated method stub
         final GLProfile profile = GLProfile.get( GLProfile.GL2 );
         GLCapabilities capabilities = new GLCapabilities( profile );
         // The canvas
         final GLCanvas glcanvas = new GLCanvas( capabilities );
         Triangledepthtest triangledepthtest = new Triangledepthtest()
         glcanvas.addGLEventListener( triangledepthtest );
         glcanvas.setSize( 400, 400 );
         final JFrame frame = new JFrame ( "3d  Triangle (solid)" );
         frame.getContentPane().add(glcanvas);
         frame.setSize( frame.getContentPane().getPreferredSize() );
         frame.setVisible( true );
         final FPSAnimator animator = new FPSAnimator( glcanvas, 300,t
         animator.start();
      }
   }
```

当编译并执行上述程序，将生成以下输出。

在这里，可以看到旋转的3D三角形的快照。由于该计划包括用于深度测试代码，三角形生成固体。




# 绘制一个3D立方体

以同样的方式，让我们得出一个3D立方体和应用颜色。通过程序来创建一个3D立方体：

```
import java.awt.DisplayMode;
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.glu.GLU;
import javax.swing.JFrame;
import com.jogamp.opengl.util.FPSAnimator;

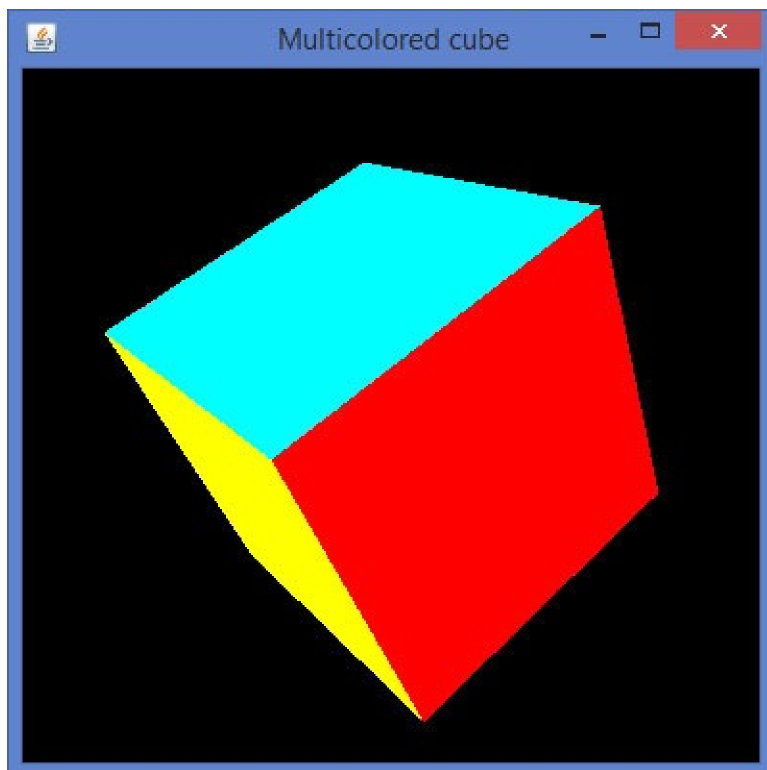
public class Cube implements GLEventListener{

   public static DisplayMode dm, dm_old;
   private GLU glu = new GLU();
   private float rquad=0.0f;
   @Override
   public void display( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glClear( GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BI
      gl.glLoadIdentity();
      gl.glTranslatef( 0f, 0f, -5.0f );
      gl.glRotatef( rquad, 1.0f, 1.0f, 1.0f ); // Rotate The Cube (
      //giving different colors to different sides
      gl.glBegin( GL2.GL_QUADS ); // Start Drawing The Cube
      gl.glColor3f( 1f,0f,0f );   //red color
      gl.glVertex3f( 1.0f, 1.0f, -1.0f ); // Top Right Of The Quad
      gl.glVertex3f( -1.0f, 1.0f, -1.0f); // Top Left Of The Quad (
      gl.glVertex3f( -1.0f, 1.0f, 1.0f ); // Bottom Left Of The Qua
      gl.glVertex3f( 1.0f, 1.0f, 1.0f ); // Bottom Right Of The Qua
```

```
      gl.glColor3f( 0f,1f,0f ); //green color
      gl.glVertex3f( 1.0f, -1.0f, 1.0f ); // Top Right Of The Quad
      gl.glVertex3f( -1.0f, -1.0f, 1.0f ); // Top Left Of The Quad
      gl.glVertex3f( -1.0f, -1.0f, -1.0f ); // Bottom Left Of The (
      gl.glVertex3f( 1.0f, -1.0f, -1.0f ); // Bottom Right Of The (
      gl.glColor3f( 0f,0f,1f ); //blue color
      gl.glVertex3f( 1.0f, 1.0f, 1.0f ); // Top Right Of The Quad (
      gl.glVertex3f( -1.0f, 1.0f, 1.0f ); // Top Left Of The Quad (
      gl.glVertex3f( -1.0f, -1.0f, 1.0f ); // Bottom Left Of The Qu
      gl.glVertex3f( 1.0f, -1.0f, 1.0f ); // Bottom Right Of The Qu
      gl.glColor3f( 1f,1f,0f ); //yellow (red + green)
      gl.glVertex3f( 1.0f, -1.0f, -1.0f ); // Bottom Left Of The Qu
      gl.glVertex3f( -1.0f, -1.0f, -1.0f ); // Bottom Right Of The
      gl.glVertex3f( -1.0f, 1.0f, -1.0f ); // Top Right Of The Quac
      gl.glVertex3f( 1.0f, 1.0f, -1.0f ); // Top Left Of The Quad (
      gl.glColor3f( 1f,0f,1f ); //purple (red + green)
      gl.glVertex3f( -1.0f, 1.0f, 1.0f ); // Top Right Of The Quad
      gl.glVertex3f( -1.0f, 1.0f, -1.0f ); // Top Left Of The Quad
      gl.glVertex3f( -1.0f, -1.0f, -1.0f ); // Bottom Left Of The (
      gl.glVertex3f( -1.0f, -1.0f, 1.0f ); // Bottom Right Of The (
      gl.glColor3f( 0f,1f, 1f ); //sky blue (blue +green)
      gl.glVertex3f( 1.0f, 1.0f, -1.0f ); // Top Right Of The Quad
      gl.glVertex3f( 1.0f, 1.0f, 1.0f ); // Top Left Of The Quad
      gl.glVertex3f( 1.0f, -1.0f, 1.0f ); // Bottom Left Of The Qua
      gl.glVertex3f( 1.0f, -1.0f, -1.0f ); // Bottom Right Of The (
      gl.glEnd(); // Done Drawing The Quad
      gl.glFlush();
      rquad -=0.15f;
   }
   @Override
   public void dispose( GLAutoDrawable drawable ) {
   // TODO Auto-generated method stub
   }
   @Override
   public void init( GLAutoDrawable drawable ) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glShadeModel( GL2.GL_SMOOTH );
      gl.glClearColor( 0f, 0f, 0f, 0f );
      gl.glClearDepth( 1.0f );
      gl.glEnable( GL2.GL_DEPTH_TEST );
      gl.glDepthFunc( GL2.GL_LEQUAL );
      gl.glHint( GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL2.GL_NICEST
   }
   @Override
   public void reshape( GLAutoDrawable drawable, int x, int y, int
      final GL2 gl = drawable.getGL().getGL2();
      if( height <=0 )
         height =1;
      final float h = ( float ) width / ( float ) height;
      gl.glViewport( 0, 0, width, height );
      gl.glMatrixMode( GL2.GL_PROJECTION );
      gl.glLoadIdentity();
      glu.gluPerspective( 45.0f, h, 1.0, 20.0 );
```

```
        gl.glMatrixMode( GL2.GL_MODELVIEW );
        gl.glLoadIdentity();
    }
    public static void main( String[] args ) {
        final GLProfile profile = GLProfile.get( GLProfile.GL2 );
        GLCapabilities capabilities = new GLCapabilities( profile );
        // The canvas
        final GLCanvas glcanvas = new GLCanvas( capabilities );
        Cube cube = new Cube();
        glcanvas.addGLEventListener( cube );
        glcanvas.setSize( 400, 400 );
        final JFrame frame = new JFrame ( " Multicolored cube" );
        frame.getContentPane().add( glcanvas );
        frame.setSize( frame.getContentPane().getPreferredSize() );
        frame.setVisible( true );
        final FPSAnimator animator = new FPSAnimator( glcanvas, 300,t
        animator.start();
    }
}
```

当编译并执行上述程序，将生成以下输出。这说明一个彩色3D立方体。




## 应用质感的立方体

给出下面的步骤来应用质感的立方体：

```
File file=new File(“c:\\pictures\\boy.jpg”);
Texture t=textureIO.newTexture(file, true);
texture=t.getTextureObject(gl);
```

- 可以要求纹理结合使用gl.glBindTexture（GL2.GL_TEXTURE_2D。纹理）绘制对象的接口方法的立方体。
- 这种方法需要的纹理（INT）参数以及GL2.GL_TEXTURE_2D（INT）
- 执行display()创建需要纹理变量。
- 在init()方法或在glDisplay()方法中的起始行，使用gl.glEnable（GL2.GL_TEXTURE_2D）方法的纹理。
- 创建纹理对象，它需要一个文件对象作为参数，而这又需要用作纹理的对象的图像的路径。
- 处理未发现异常文件。

让我们通过程序应用纹理到多维数据集：

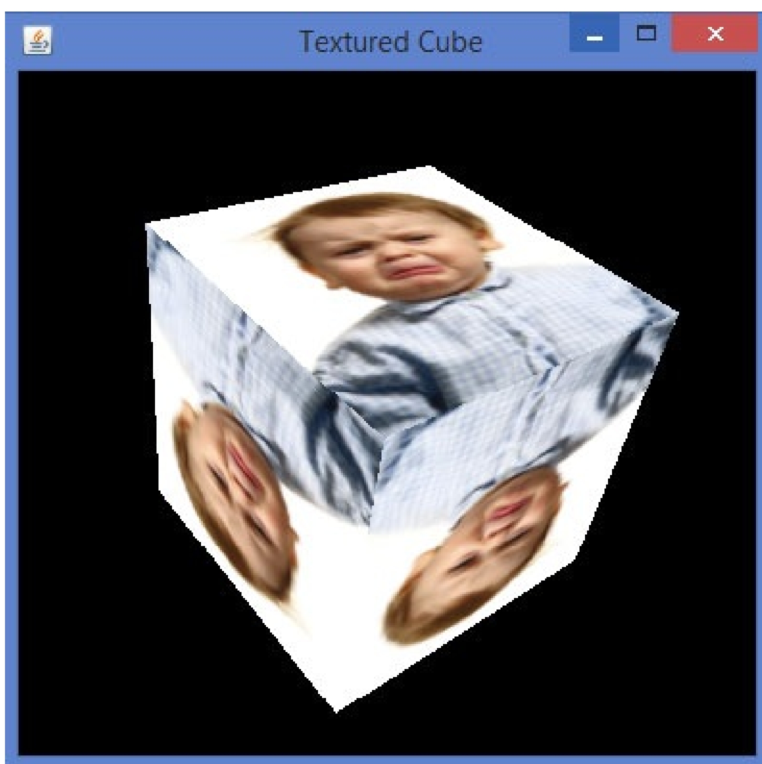
```
import java.awt.DisplayMode;
import java.io.File;
import java.io.IOException;
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.glu.GLU;
import javax.swing.JFrame;
import com.jogamp.opengl.util.FPSAnimator;
import com.jogamp.opengl.util.texture.Texture;
import com.jogamp.opengl.util.texture.TextureIO;

public class CubeTexture implements GLEventListener {
   public static DisplayMode dm, dm_old;
   private GLU glu = new GLU();
   private float xrot,yrot,zrot;
   private int texture;
   @Override
   public void display(GLAutoDrawable drawable) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT
      gl.glLoadIdentity();                       // Reset The View
      gl.glTranslatef(0f, 0f, -5.0f);
      gl.glRotatef(xrot, 1.0f, 1.0f, 1.0f);
      gl.glRotatef(yrot, 0.0f, 1.0f, 0.0f);
      gl.glRotatef(zrot, 0.0f, 0.0f, 1.0f);
      gl.glBindTexture(GL2.GL_TEXTURE_2D, texture);
```

```
      gl.glBegin(GL2.GL_QUADS);
      // Front Face
      gl.glTexCoord2f(0.0f, 0.0f); gl.glVertex3f(-1.0f, -1.0f,  1.0
      gl.glTexCoord2f(1.0f, 0.0f); gl.glVertex3f( 1.0f, -1.0f,  1.0
      gl.glTexCoord2f(1.0f, 1.0f); gl.glVertex3f( 1.0f,  1.0f,  1.0
      gl.glTexCoord2f(0.0f, 1.0f); gl.glVertex3f(-1.0f,  1.0f,  1.0
      // Back Face
      gl.glTexCoord2f(1.0f, 0.0f); gl.glVertex3f(-1.0f, -1.0f, -1.0
      gl.glTexCoord2f(1.0f, 1.0f); gl.glVertex3f(-1.0f,  1.0f, -1.0
      gl.glTexCoord2f(0.0f, 1.0f); gl.glVertex3f( 1.0f,  1.0f, -1.0
      gl.glTexCoord2f(0.0f, 0.0f); gl.glVertex3f( 1.0f, -1.0f, -1.0
      // Top Face
      gl.glTexCoord2f(0.0f, 1.0f); gl.glVertex3f(-1.0f,  1.0f, -1.0
      gl.glTexCoord2f(0.0f, 0.0f); gl.glVertex3f(-1.0f,  1.0f,  1.0
      gl.glTexCoord2f(1.0f, 0.0f); gl.glVertex3f( 1.0f,  1.0f,  1.0
      gl.glTexCoord2f(1.0f, 1.0f); gl.glVertex3f( 1.0f,  1.0f, -1.0
      // Bottom Face
      gl.glTexCoord2f(1.0f, 1.0f); gl.glVertex3f(-1.0f, -1.0f, -1.0
      gl.glTexCoord2f(0.0f, 1.0f); gl.glVertex3f( 1.0f, -1.0f, -1.0
      gl.glTexCoord2f(0.0f, 0.0f); gl.glVertex3f( 1.0f, -1.0f,  1.0
      gl.glTexCoord2f(1.0f, 0.0f); gl.glVertex3f(-1.0f, -1.0f,  1.0
      // Right face
      gl.glTexCoord2f(1.0f, 0.0f); gl.glVertex3f( 1.0f, -1.0f, -1.0
      gl.glTexCoord2f(1.0f, 1.0f); gl.glVertex3f( 1.0f,  1.0f, -1.0
      gl.glTexCoord2f(0.0f, 1.0f); gl.glVertex3f( 1.0f,  1.0f,  1.0
      gl.glTexCoord2f(0.0f, 0.0f); gl.glVertex3f( 1.0f, -1.0f,  1.0
      // Left Face
      gl.glTexCoord2f(0.0f, 0.0f); gl.glVertex3f(-1.0f, -1.0f, -1.0
      gl.glTexCoord2f(1.0f, 0.0f); gl.glVertex3f(-1.0f, -1.0f,  1.0
      gl.glTexCoord2f(1.0f, 1.0f); gl.glVertex3f(-1.0f,  1.0f,  1.0
      gl.glTexCoord2f(0.0f, 1.0f); gl.glVertex3f(-1.0f,  1.0f, -1.0
      gl.glEnd();
      gl.glFlush();
      //change the speeds here
      xrot+=.1f;
      yrot+=.1f;
      zrot+=.1f;
   }
   @Override
   public void dispose(GLAutoDrawable drawable) {
      // method body
   }
   @Override
   public void init(GLAutoDrawable drawable) {
      final GL2 gl = drawable.getGL().getGL2();
      gl.glShadeModel(GL2.GL_SMOOTH);
      gl.glClearColor(0f, 0f, 0f, 0f);
      gl.glClearDepth(1.0f);
      gl.glEnable(GL2.GL_DEPTH_TEST);
      gl.glDepthFunc(GL2.GL_LEQUAL);
      gl.glHint(GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL2.GL_NICEST);
      gl.glEnable(GL2.GL_TEXTURE_2D);
      try
```

```
        {
           File im = new File("E:\\office\\boy.jpg ");
           Texture t = TextureIO.newTexture(im, true);
           texture= t.getTextureObject(gl);
        }
        catch(IOException e)
        {
           e.printStackTrace();
        }
     }
     @Override
     public void reshape(GLAutoDrawable drawable, int x, int y, int w
        final GL2 gl = drawable.getGL().getGL2();
        if(height <=0)
           height =1;
        final float h = (float) width / (float) height;
        gl.glViewport(0, 0, width, height);
        gl.glMatrixMode(GL2.GL_PROJECTION);
        gl.glLoadIdentity();
        glu.gluPerspective(45.0f, h, 1.0, 20.0);
        gl.glMatrixMode(GL2.GL_MODELVIEW);
        gl.glLoadIdentity();
     }
     public static void main(String[] args) {
        final GLProfile profile = GLProfile.get(GLProfile.GL2);
        GLCapabilities capabilities = new GLCapabilities(profile);
        // The canvas
        final GLCanvas glcanvas = new GLCanvas(capabilities);
        CubeTexture r = new CubeTexture();
        glcanvas.addGLEventListener(r);
        glcanvas.setSize(400, 400);
        final JFrame frame = new JFrame (" Textured Cube");
        frame.getContentPane().add(glcanvas);
        frame.setSize(frame.getContentPane().getPreferredSize());
        frame.setVisible(true);
        final FPSAnimator animator = new FPSAnimator(glcanvas, 300,tr
        animator.start();
     }
  }
```

当编译并执行上述程序，将生成以下输出。可以看到一个3D立方体与应用上所需的纹理。




# 附录

GPU - 图形处理单元，它是促进图像的呈现一个特殊的电子设备。

JNI - Java本地接口。使用它，Java可以访问本地方法。

模型 - 它们是从基本的图形元素，如点，线和多边形构造的对象。

像素 - 显示在屏幕上看到的最小单位。

投影 - 映射对象的坐标的二维平面的方法，被称为凸起。

投影矩阵 - 它是二维表面上的物体的线性变换。

渲染 - 由计算机从模型生成的图像的过程。

视口 - 视区是计算机图形学的屏幕上观看区域。

# JPA教程

任何企业应用程序通过存储和检索大量数据进行数据库操作。尽管所有的存储管理提供技术，应用程序开发人员通常很难有效地执行数据库操作。

一般情况下，Java开发人员使用大量的代码，或使用专有的架构与数据库进行交互，而使用JPA与数据库绑定交互负担显著降低。它形成（数据库程序）对象模型之间的桥梁（Java程序）和关系模型。

## 关系型和对象模型之间的不匹配

关系对象表示以表格的形式，而对象模型表示的对象格式的相互连接的图形。而存储和检索来自关系数据库的对象模型，一些不匹配的发生是由于以下原因：

- 粒度：对象模型比关系模型更精细。
- 亚型：亚型（指继承）所有类型的关系数据库不支持。
- 标识：如对象模型，关系模型并没有同时编写暴露身份。
- 关联：关系模型无法确定多重关系，同时寻找到一个对象域模型。
- 数据导航：在一个对象网络对象之间的数据导航是在这两种模式中有所不同。

## JPA是什么?

Java持久性API(简称JAP)是类和方法的集合，以海量数据关系映射持久并存储到数据库，这是由Oracle公司提供方案技术。

## 在哪里使用JPA？

为了减少编写代码，对象关系管理的负担，程序员遵循“JPA提供者”框架，它可以方便地与数据库实例的交互。这里所需要的框架接管JPA。



## JPA 历史

早期版本的EJB，定义持久层结合使用 javax.ejb.EntityBean 接口作为业务逻辑层。

- 同时引入EJB3.0的持久层分离，并指定为JPA1.0（Java持久性API）。这个API规范随着JAVA EE5对2006年5月11日使用JSR220规范发布。

- JPA2.0的JAVA EE 6规范发布于2009年12月10日并成Java Community Process JSR317 的一部分。
- JPA2.1使用JSR338 的 JAVA EE7的规范发布于2013年4月22日。

## JPA提供者

JPA是一个开源的API，因此各企业厂商如Oracle，Redhat，Eclipse等，通过增加JPA 持续性，在提供JPA的新产品。这些产品包括：

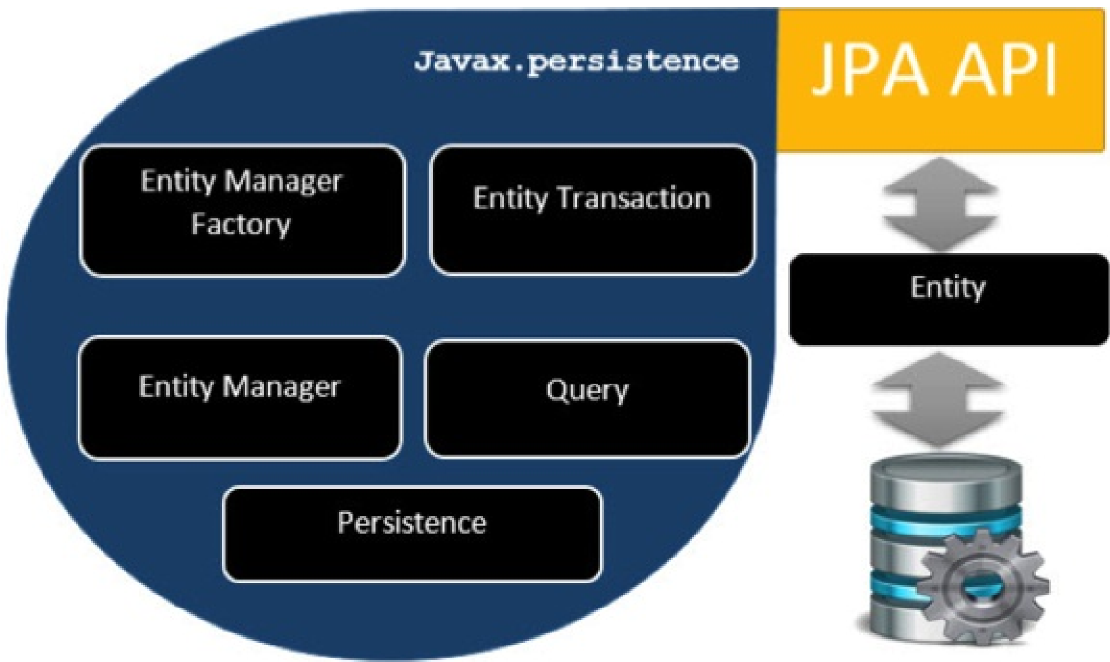
Hibernate, Eclipselink, Toplink, Spring Data JPA, etc.

# JPA架构 - JPA教程

JPA(Java持久性API)是存储业务实体关联的实体的来源。它显示了如何定义一个面向普通Java对象(POJO)作为一个实体，以及如何与管理关系实体。

## 类级别架构

下图显示了JPA的类的层次结构。它显示核心类和JPA接口。



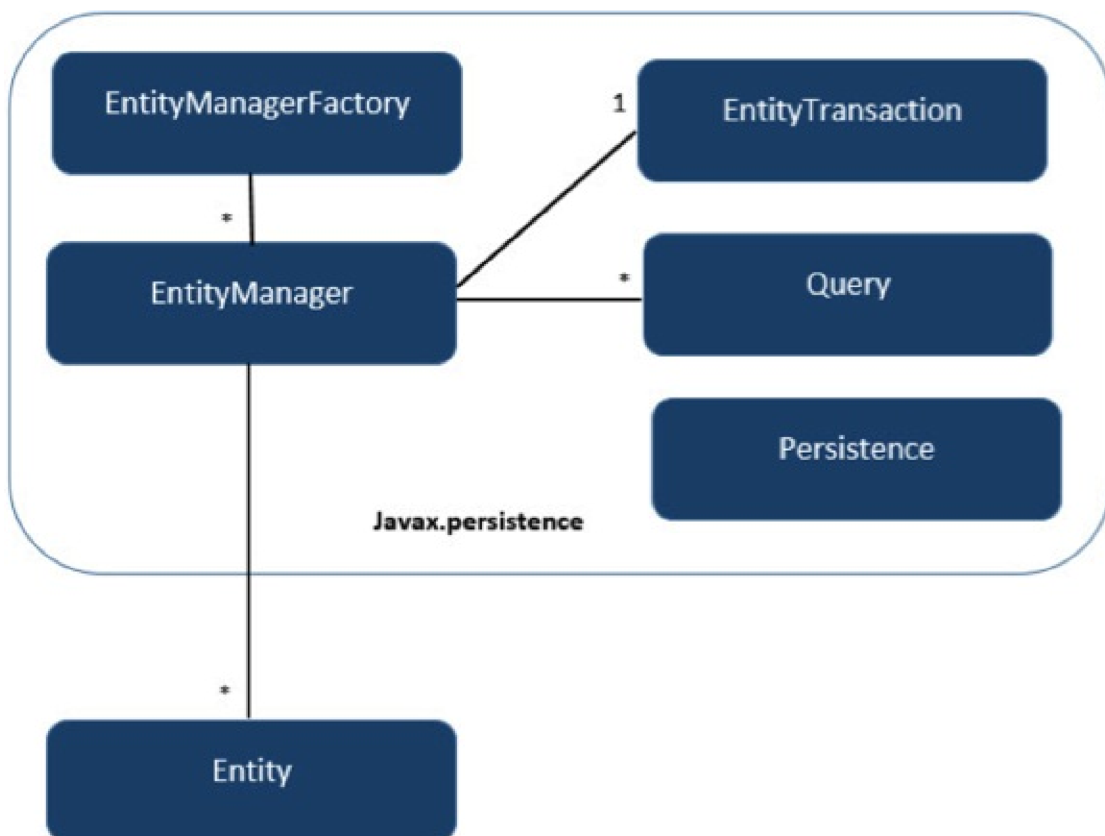

下表描述了每个在上述架构的显示单元。

| 单元 | 描述 |
| --- | --- |
| **EntityManagerFactory** | 这是一个EntityManager的工厂类。它创建并管理多个EntityManager实例。 |
| **EntityManager** | 这是一个接口，它管理的持久化操作的对象。它的工作原理类似工厂的查询实例。 |
| **Entity** | 实体是持久性对象是存储在数据库中的记录。 |
| **EntityTransaction** | 它与EntityManager是一对一的关系。对于每一个EntityManager，操作是由EntityTransaction类维护。 |
| **Persistence** | 这个类包含静态方法来获取EntityManagerFactory实例。 |
| **Query** | 该接口由每个JPA供应商，能够获得符合标准的关系对象。 |

上述的类和接口用于存储实体到数据库的一个记录。帮助程序员通过减少自己编写代码将数据存储到数据库中，使他们能够专注于更重要的业务活动代码，如 数据库表映射的类编写代码。

## JPA类关系

在上述体系结构中，类和接口之间的关系属于javax.persistence包。下图显示了它们之间的关系。




- EntityManagerFactory和EntityManager的关系是1对多。这是一个工厂类EntityManager实例。
- EntityManager和EntityTransaction之间的关系是1对1。对于每个EntityManager操作，只有一个EntityTransaction实例。
- EntityManager和Query之间的关系是1对多。查询数众多可以使用一个EntityManager实例执行。
- EntityManager实体之间的关系是1对多。一个EntityManager实例可以管理多个实体。

# JPA ORM组件 - JPA教程

最现代的应用程序使用关系型数据库来存储数据。最近，许多厂商改用对象数据库，以减少其对数据的维护负担。这意味着对象数据库或对象关系技术正在存储，检索，更新和维护数据的照顾。这个对象关系型技术的核心部分是映射orm.xml中的文件。随着XML不需要编译，可以很容易地进行修改多个数据源较少的管理。

## 对象关系映射

对象关系映射(ORM)简要地告诉什么是ORM以及它是如何工作。 ORM是从对象类型的数据隐蔽到关系型，反之亦然编程能力。

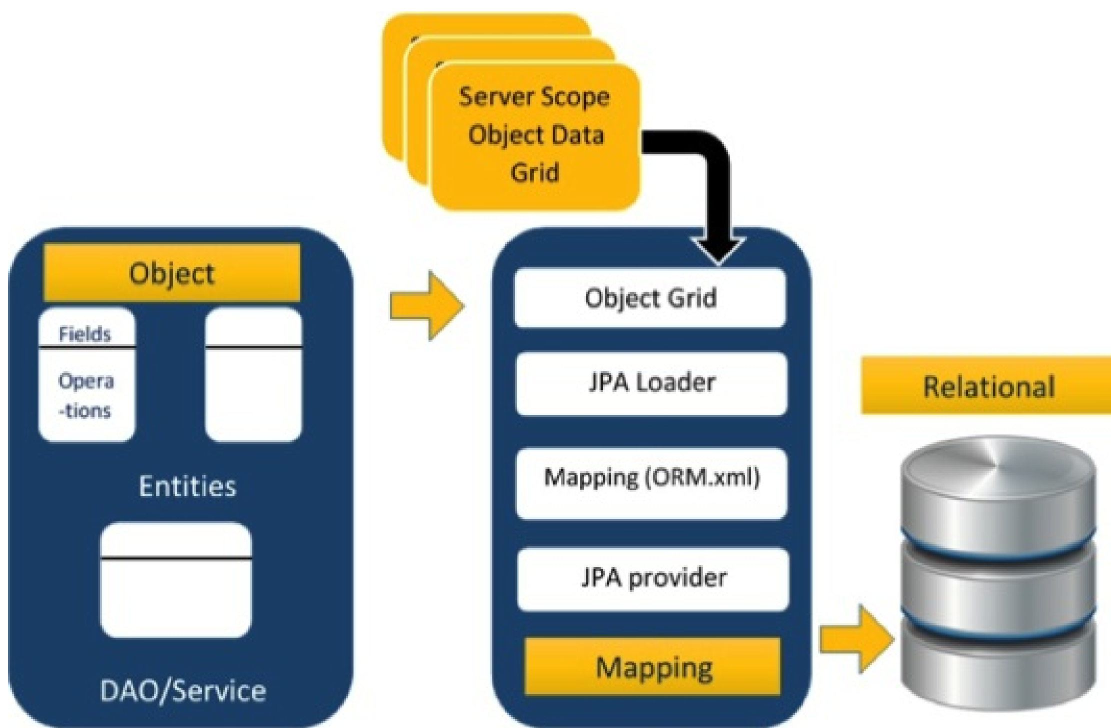
ORM主要特征是映射或绑定一个目的是它的数据库中的数据。而映射，我们要考虑的任何其他表中的数据，数据的类型，并具有自一个或多个实体的关系。

## 高级功能

- 惯用的持久性：它使您能够编写使用面向对象的类持久性类。
- 高性能：它有许多抓取技术和充满希望的锁定技术。
- 可靠的：它是高度稳定的，被很多专业程序员。

## ORM架构

在ORM架构如下所示。




在上述体系结构解释了如何对象数据存储到关系数据库中的三个阶段。

## 第1阶段

第一阶段，命名为对象数据阶段，包括POJO类，服务接口和类。它是主要的业务组件层，其具有业务逻辑操作和属性。

例如，让我们举个员工数据库的架构。

- Employee POJO类包含属性，如ID，姓名，工资和标识。它也包含类似属性setter和getter方法。
- Employee DAO/服务类包含服务方法，如建立员工，发现员工和删除员工。

## 第2阶段

第二阶段，称为映射或持久性的阶段，包括JPA提供者，映射文件（orm.xml），JPA装载器和对象网格。

- JPA提供者：这是一个包含了JPA（javax.persistence）供应的产品。例如EclipseLink，Toplink，Hibernate等。
- 映射文件：映射文件（orm.xml中）包含在关系数据库中的一个POJO类的数据和数据之间的映射配置。
- JPA装载器：在JPA加载器的工作原理就像一个高速缓冲存储器。它可以加载关系网格数据。它的工作原理类似数据库的副本与服务类POJO数据（POJO类的属性）进行交互。

- 对象网格：它是可存储的关系数据的副本，如高速缓冲存储器的临时位置。对数据库的所有查询首先被实现在对象网格的数据。只有提交它才会影响到主数据库。

### 第3阶段

第三阶段是关系数据相关。它包含在逻辑上连接到所述业务组件的关系数据。如上所讨论的，仅当业务组件提交该数据，它被存储到数据库中的物理。在此之前，已修改的数据被存储在高速缓冲存储器作为一个网格格式。在获取数据的过程和存储数据是相同的。

上述三个阶段的编程交互的机制被称为对象关系映射。

## Mapping.xml

mapping.xml文件指示JPA的供应者来映射实体类与数据库表。

让我们以Employee实体包含四个属性的一个例子。POJO类Employee实体的命名为：Employee.java，如下：

```
public class Employee
{
   private int eid;
   private String ename;
   private double salary;
   private String deg;
   public Employee(int eid, String ename, double salary, String deg
   {
       super( );
       this.eid = eid;
       this.ename = ename;
       this.salary = salary;
       this.deg = deg;
   }

   public Employee( )
   {
       super();
   }

   public int getEid( )
   {
       return eid;
   }
   public void setEid(int eid)
   {
       this.eid = eid;
   }
    public String getEname( )
```

```
    {
        return ename;
    }
    public void setEname(String ename)
    {
        this.ename = ename;
    }

    public double getSalary( )
    {
        return salary;
    }
    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    public String getDeg( )
    {
        return deg;
    }
    public void setDeg(String deg)
    {
        this.deg = deg;
    }
}
```

上面的代码是Employee实体POJO类。它包含四个属性eid, ename,salary, 和deg。考虑这些属性为表的字段，并且eid作为该表的主键。现在，我们要设计Hibernate映射文件了。映射文件名为 mapping.xml 如下：

```
<? xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm'
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
                        http://java.sun.com/xml/ns/persistence/orm_
                        version="1.0">
    <description> XML Mapping file</description>
    <entity class="Employee">
        <table name="EMPLOYEETABLE"/>
        <attributes>
            <id name="eid">
                <generated-value strategy="TABLE"/>
            </id>
            <basic name="ename">
                <column name="EMP_NAME" length="100"/>
            </basic>
            <basic name="salary">
            </basic>
            <basic name="deg">
            </basic>
        </attributes>
    </entity>
</entity-mappings>
```

上述脚本用于与数据库表的映射实体类。在该文件中

- <entity-mappings> : 标签定义的模式定义，允许实体标记为XML文件。
- <description> : 标签提供了有关应用程序的描述。
- <entity> : 标签定义要转换成数据库表中的实体类。属性类定义了POJO实体类的名称。
- <table> : 标签定义的表名。如果想有两个类相同的名称以及该表中，则该标签是没有必要的。
- <attributes> : 标签定义的属性（在表中的字段）。
- <id> : 标记定义表中的主键。在<generated-value>标记定义了如何将主键值赋值，如Automatic, Manual或者使用 Sequence。
- <basic> : 标签用于定义其余属性在表中。
- <column-name> : 标签被用来在表中定义用户定义表的字段名。

# 注解

一般的XML文件用于配置特定的组件，或者映射两种不同规格的组件。在我们的例子中，我们要分别保持在一个框架的XML文件。这意味着在写一个映射的XML文件，我们需要比较用mapping.xml文件实体标签的POJO类的属性。

这里是解决方案。在类定义中，我们可以使用注释写配置的一部分。注解用于类，属性和方法。注释以'@'符号在类，属性或方法的注释中声明之前。 JPA的所有批注在javax.persistence包定义。

在这里，在我们的实例中使用的注释列表如下。

| 注解 | 描述 |
|---|---|
| @Entity | 声明类为实体或表。 |
| @Table | 声明表名。 |
| @Basic | 指定非约束明确的各个字段。 |
| @Embedded | 指定类或它的值是一个可嵌入的类的实例的实体的属性。 |
| @Id | 指定的类的属性，用于识别（一个表中的主键）。 |
| @GeneratedValue | 指定如何标识属性可以被初始化，例如自动，手动，或从序列表中获得的值。 |
| @Transient | 指定的属性，它是不持久的，即，该值永远不会存储在数据库中。 |
| @Column | 指定持久属性栏属性。 |
| @SequenceGenerator | 指定在@GeneratedValue注解中指定的属性的值。它创建了一个序列。 |
| @TableGenerator | 指定在@GeneratedValue批注指定属性的值发生器。它创造了的值生成的表。 |
| @AccessType | 这种类型的注释用于设置访问类型。如果设置@AccessType（FIELD），然后进入FIELD明智的。如果设置@AccessType（PROPERTY），然后进入属性发生明智的。 |
| @JoinColumn | 指定一个实体组织或实体的集合。这是用在多对一和一对多关联。 |
| @UniqueConstraint | 指定的字段和用于主要或辅助表的唯一约束。 |
| @ColumnResult | 参考使用select子句的SQL查询中的列名。 |
| @ManyToMany | 定义了连接表之间的多对多一对多的关系。 |
| @ManyToOne | 定义了连接表之间的多对一的关系。 |
| @OneToMany | 定义了连接表之间存在一个一对多的关系。 |
| @OneToOne | 定义了连接表之间有一个一对一的关系。 |
| @NamedQueries | 指定命名查询的列表。 |
| @NamedQuery | 指定使用静态名称的查询。 |

# Java Bean标准

Java类封装了实例的值及其行为为对象称为一个单元。 Java Bean是一个临时的存储和可重用的组件或对象。它是有一个默认的构造函数和getter和setter方法来初始化实例序列化的类单独的属性。

## Bean约定

- bean包含其默认构造函数或包含序列化实例的文件。因此，一个bean可以实例化另一个bean。
- bean属性可以被隔离成布尔属性或者非布尔属性。
- 非布尔属性包含getter和setter方法。
- 布尔属性包含setter和方法。
- 任何字段的getter方法应从小字母get（Java方法的公约）开始，之后使用大写字母开头的字段名。例如，字段名为salary，因此这一字段的getter方法为getSalary()。
- 任何属性的setter方法应该先从小字母的集合（Java方法公约）开始，继续以大写字母，参数值设置为字段开头的字段名。例如，字段名为salary，因此这一字段的setter方法是setSalary（double sal ）。
- 对于布尔型属性，方法是检查它是否是 true 或 false。例如，Boolean属性为空，则该字段的就是方法isEmpty()。

# JPA安装配置 - JPA教程

本章将指导完成JPA在Windows和Linux系统的设置过程。 JPA可以很容易地安装并集成而无需任何复杂的设置程序，简单几个步骤在当前的Java环境。在安装时用户管理是必需的。

## 系统要求

| JDK | Java SE 2 JDK 1.5 或以上 |
|---|---|
| 内存 | 1 GB RAM (推荐，小了慢了不要怪我) |
| 硬盘 | 没有最小要求 |
| 系统版本 | Windows XP 或以上, Linux |

现在让我们继续安装JPA，如下几个步骤。

## 第一步：确认已经Java安装

首先，需要在系统上安装Java软件开发工具包（SDK）。为了验证这一点，根据所使用的平台执行以下两个命令。

如果Java安装已正确完成，那么它会显示Java安装的当前版本。示例输出如下表中。

### Windows

打开命令控制台，然后输入：

```
>java –version
```

```
Java version "1.7.0_60"
Java (TM) SE Run Time Environment (build 1.7.0_60-b19)
Java Hotspot (TM) 64-bit Server VM (build 24.60-b09,mixed mode)
```

### Linux

打开命令终端，输入：

```
$java –version
```

```
java version "1.7.0_25"
Open JDK Runtime Environment (rhel-2.3.10.4.el6_4-x86_64)
Open JDK 64-Bit Server VM (build 23.7-b01, mixed mode)
```

- 假设本教程的读者已把Java SDK版本1.7.0_60 安装在他们的系统中。
- 如果还没有安装Java SDK，从 http://www.oracle.com/technetwork/java/javase/downloads/index.html 下载当前版本并安装它。

## 第2步：设置Java环境

设置环境变量JAVA_HOME 指向到安装在机器上的 Java 目录的位置。例如，

| 平台 | 描述 |
| --- | --- |
| Windows | 设置 JAVA_HOME 指向 C:\ProgramFiles\java\jdk1.7.0_60 |
| Linux | Export JAVA_HOME=/usr/local/java-current |

添加Java编译器位置的完整路径到系统路径。

| 平台 | 描述 |
| --- | --- |
| Windows | 添加字符串 "C:\Program Files\Java\jdk1.7.0_60\bin" 到系统变量 PATH 的尾部. |
| Linux | Export PATH=$PATH:$JAVA_HOME/bin/ |

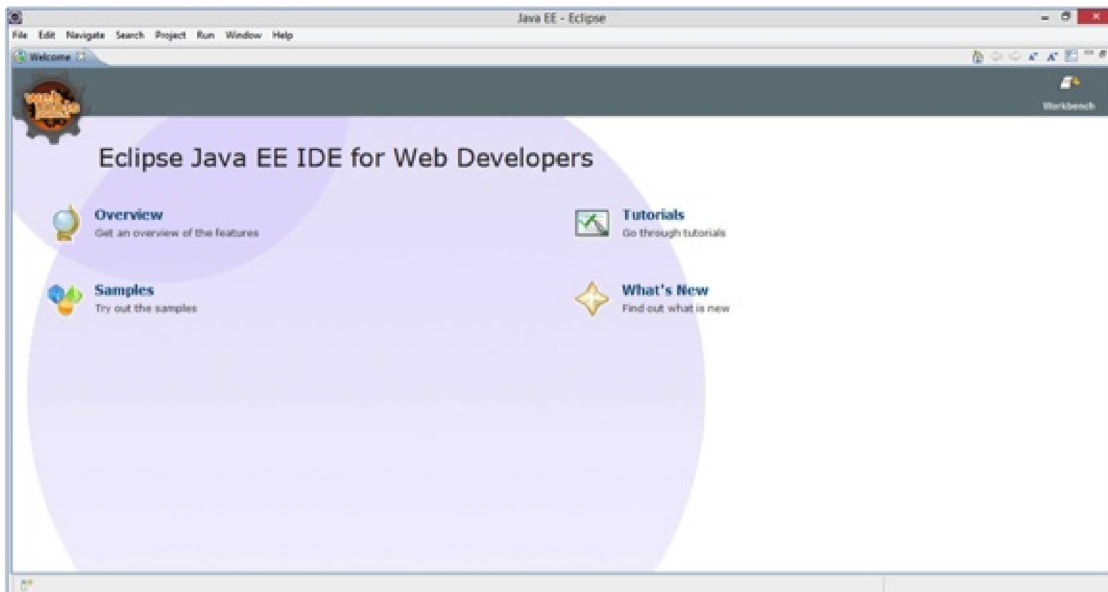
在命令提示符下执行命令java-version如上所述。

## 第3步：安装JPA

可以使用任何JPA提供者，从本教程中，如EclipseLink，Hibernate都经过JPA安装。让我们使用EclipseLink遵循JPA安装。对于JPA编程需要遵循特定的文件夹架构，因此最好是使用IDE。

下载Eclipse IDE 如下面的链接 https://www.eclipse.org/downloads/ 选择 EclipseIDE 对JavaEE 开发者是 Eclipse indigo。

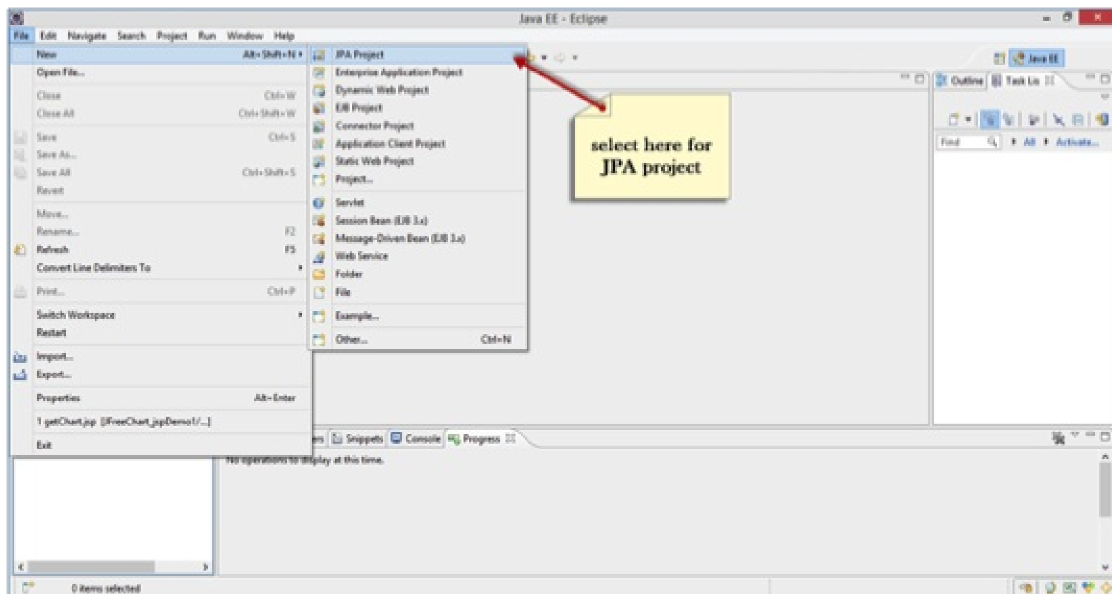
解压缩 Eclipse 的zip文件。打开Eclipse IDE。



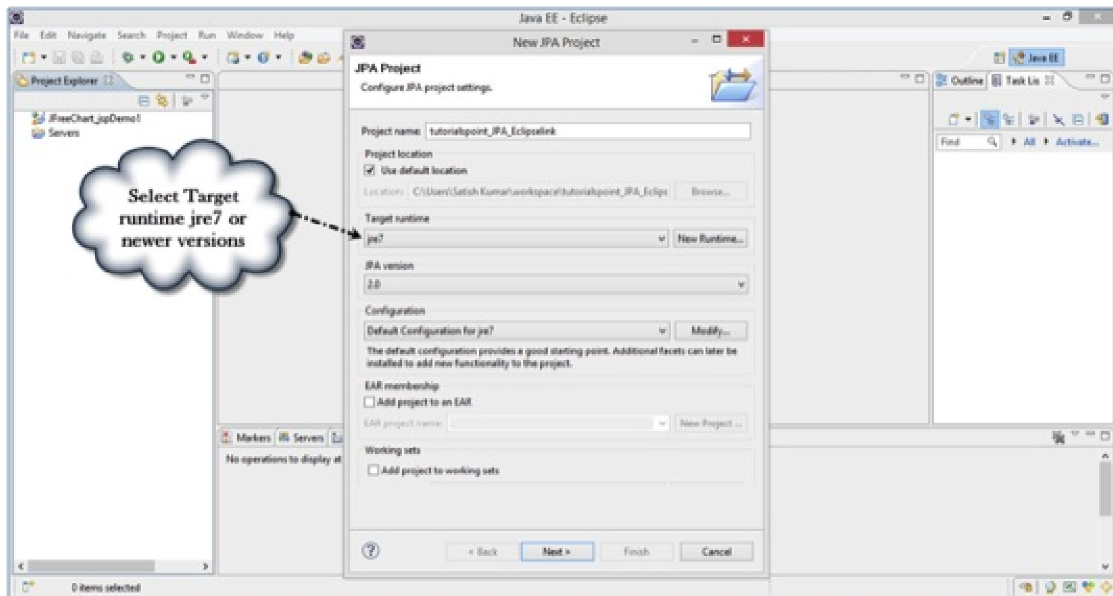

## 安装使用EclipseLink JPA

EclipseLink是一个库，因此我们不能直接将其添加到Eclipse IDE。安装JPA使用EclipseLink需要按照下面给出的步骤。

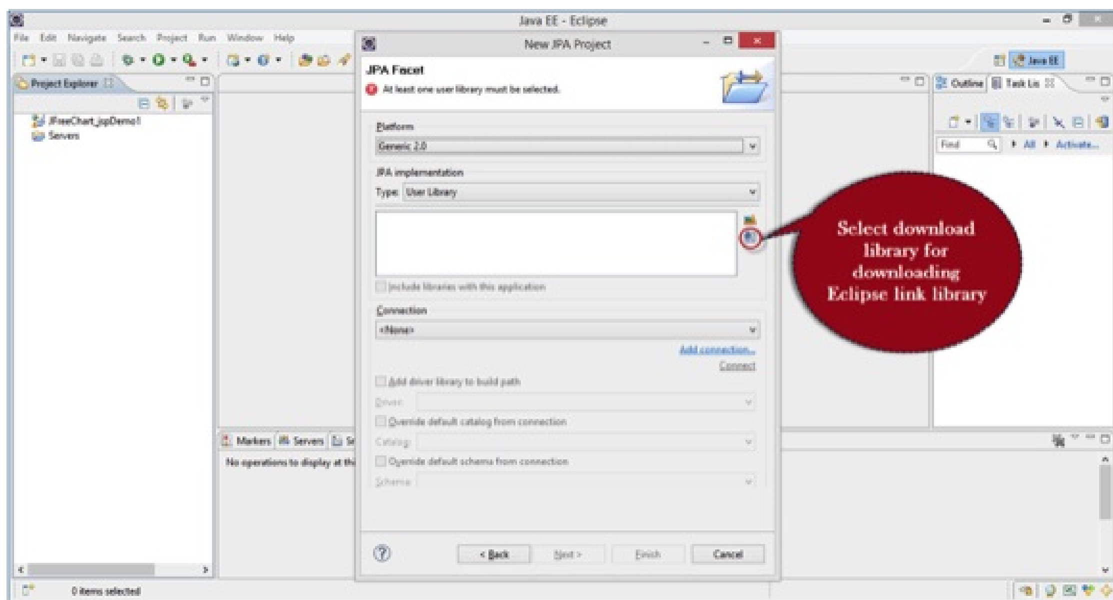
- 创建一个新的JPA项目在Eclipse IDE中，首先选择File->New->JPA项目如下：



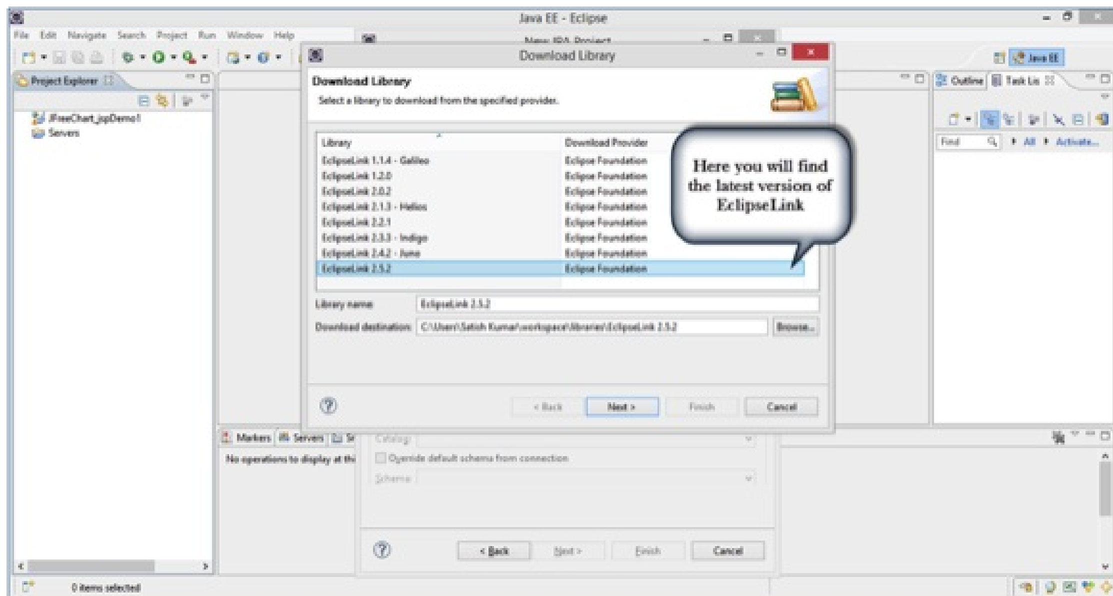

- 得到一个名为新建JPA项目的对话框。输入项目名称 yiibai_JPA_Eclipselink，检查JRE版本，然后单击下一步：



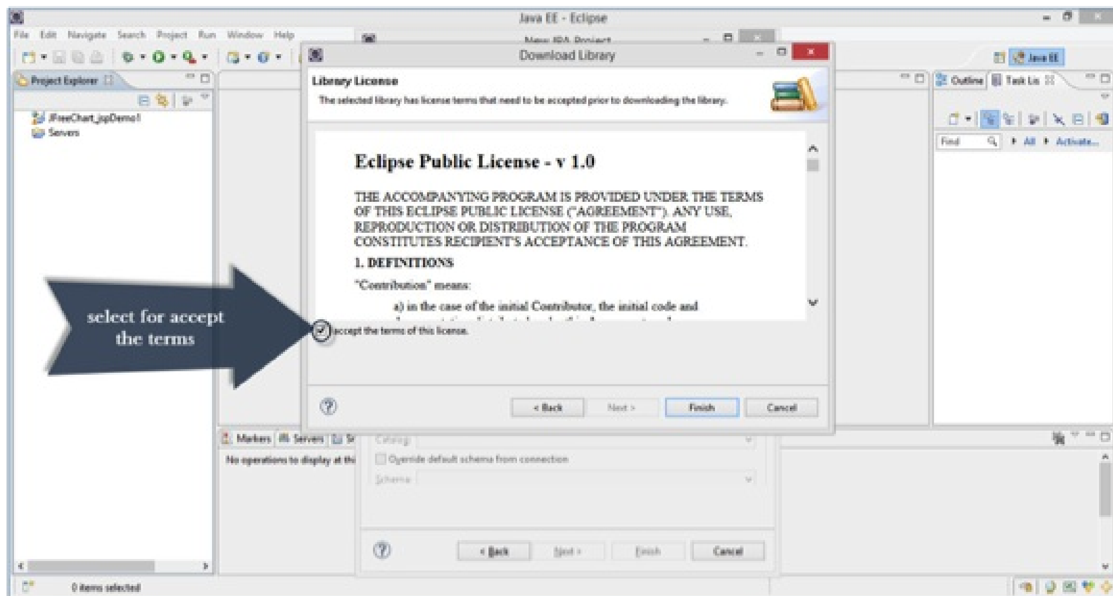

- 点击下载库（如果没有库）中的用户库部分。



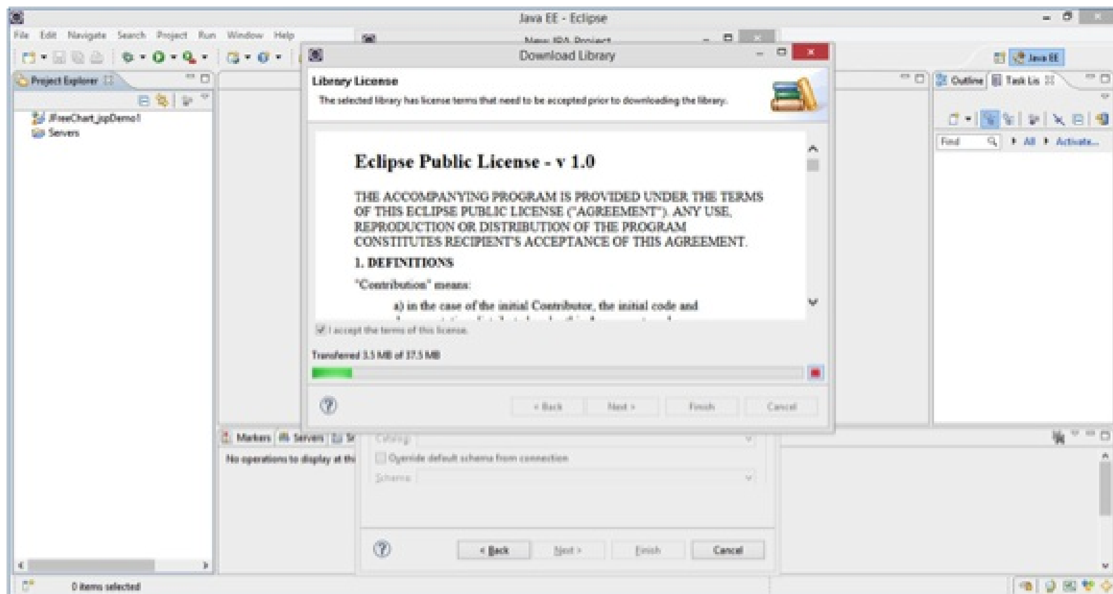

- 在下载库对话框中选择 EclipseLink 库的最新版本，点击下一步，如下所示：



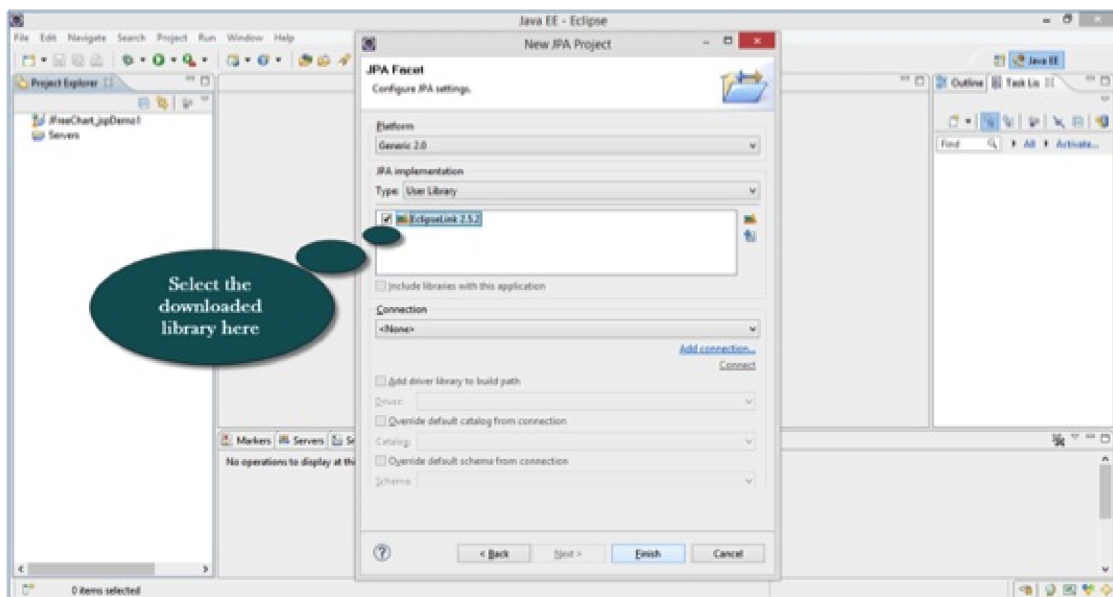

- 接受许可条款，然后单击下载库完成。



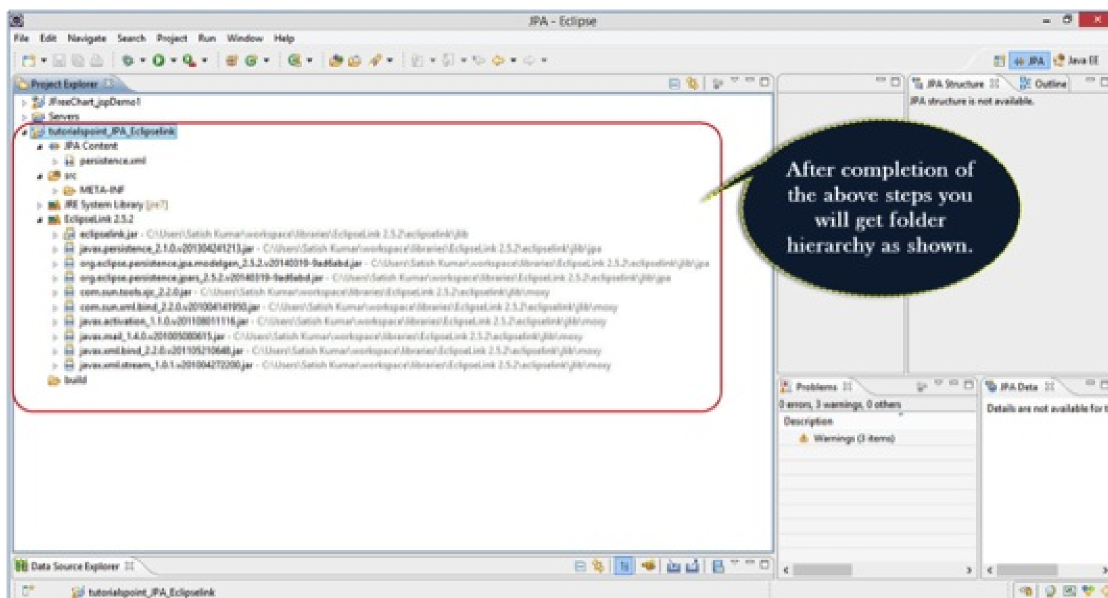

- 下载开始作为显示在下面的屏幕截图。




- 下载后，请在用户库部分中的下载库，然后单击Finish（完成）。




- 最后得到Package Explorer中的Eclipse IDE项目文件。提取所有文件，得到的文件夹和文件的层次结构如下所示：

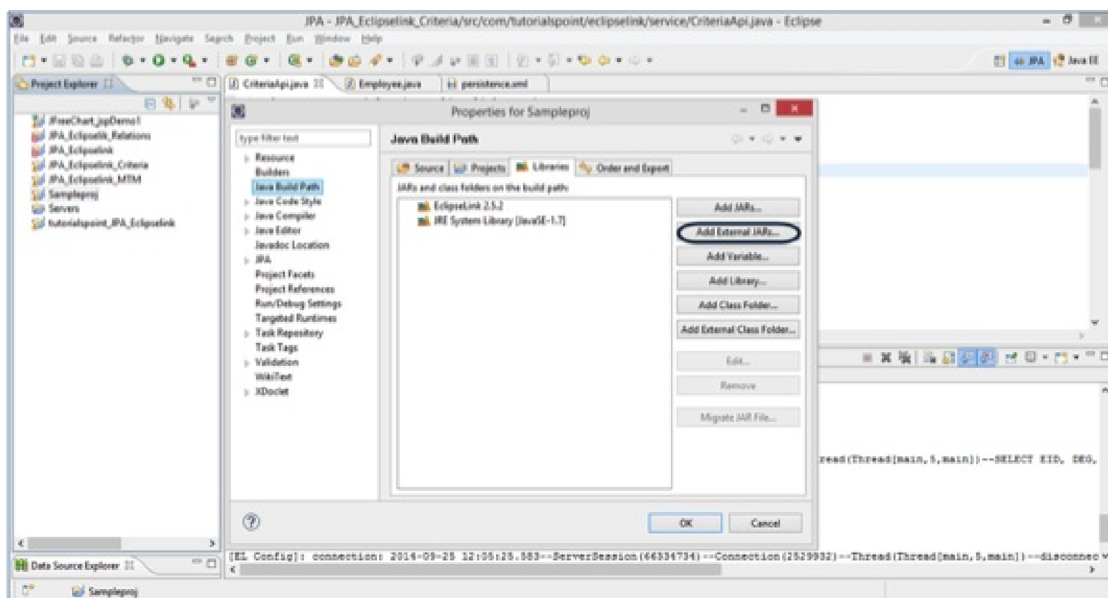



# 加入MySQL连接到项目

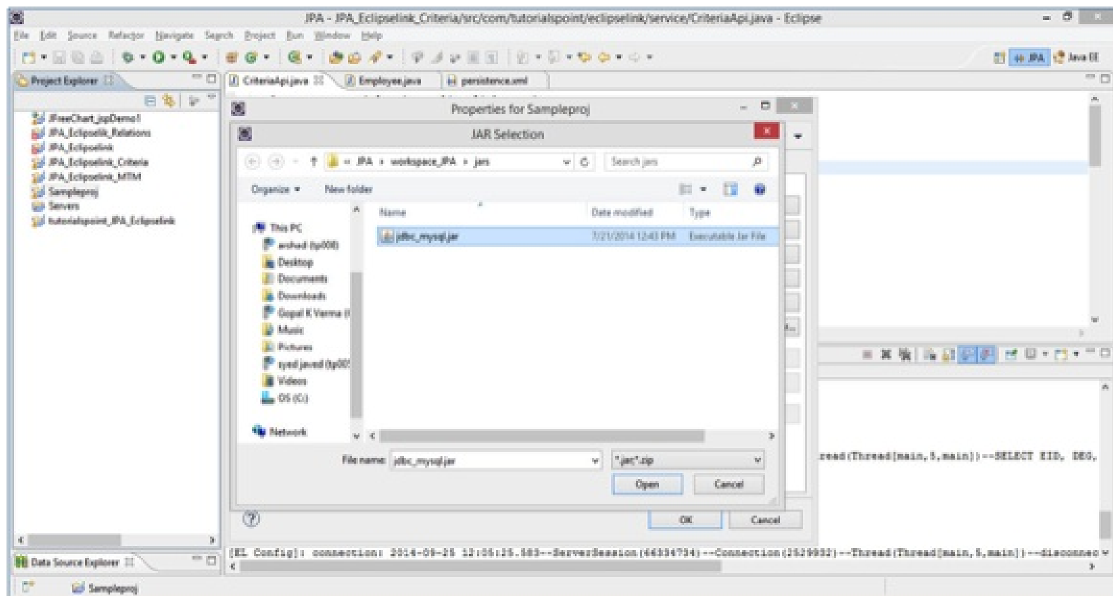
我们在这里讨论的例子需要连接数据库。让我们看看MySQL数据库进行数据库操作。它需要使用mysql-connector jar与Java程序进行交互。

按照以下步骤配置数据库的jar在项目中。

- 转到项目属性 - 通过右击它> Java构建路径。会得到一个对话框，显示在下面的屏幕截图。单击添加外部JAR。



- 去到 jar 在系统的位置，选择该文件，然后单击打开。




- 单击确定在属性对话框。将获得MySQL的连接器jar 在项目。现在，可以使用MySQL数据库操作。

# JPA实体管理器 - JPA教程

本章将使用一个简单的例子来说明JPA是如何工作的。让我们来考虑以员工管理为例。假设员工管理分别创建，更新，查找和删除员工的记录操作。正如前面提到的，使用MySQL数据库进行数据库操作。

对于此示例中的主要模块如下：

- 模型或POJO

  Employee.java

- 持久化

  Persistence.xml

- 服务

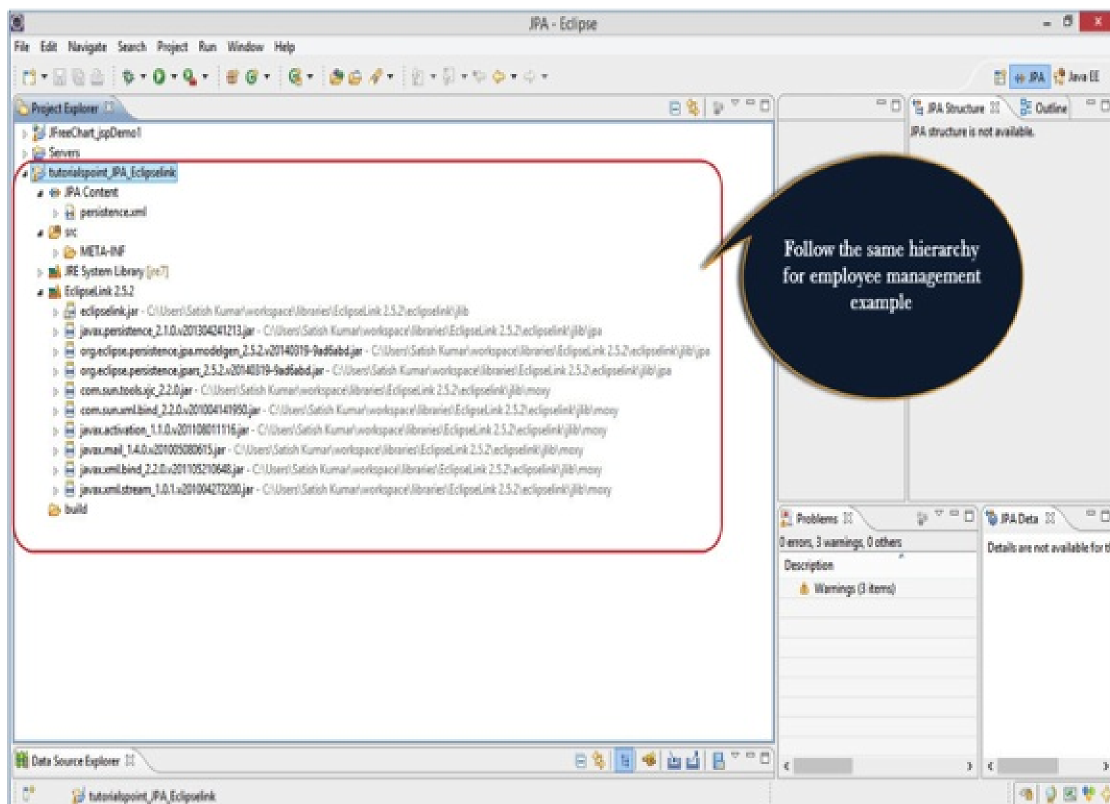
  CreatingEmployee.java

  UpdatingEmployee.java

  FindingEmployee.java

  DeletingEmployee.java

让我们看看，我们已经使用了JPA安装EclipseLink包的层次。按照分层结构的此示例中，如下所示：



# 创建实体

实体是什么？无非是bean或model。在这个例子中，我们将使用员工作为一个实体。 eid, ename, salary, 和deg是实体的属性。它包含一个默认的构造函数，以及这些属性的setter和getter方法。

在上图所示的层次结构，创建一个名为“com.yiibai.eclipselink.entity'包，在'src'中(源)封装。给包下创建一个名为Employee.java类，如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table
public class Employee
{
   @Id
   @GeneratedValue(strategy= GenerationType.AUTO)
   private int eid;
   private String ename;
   private double salary;
   private String deg;
   public Employee(int eid, String ename, double salary, String deg
   {
       super( );
       this.eid = eid;
       this.ename = ename;
       this.salary = salary;
       this.deg = deg;
   }

   public Employee( )
   {
       super();
   }

   public int getEid( )
   {
       return eid;
   }
   public void setEid(int eid)
   {
       this.eid = eid;
   }
    public String getEname( )
   {
```

```
        return ename;
    }
    public void setEname(String ename)
    {
        this.ename = ename;
    }

    public double getSalary( )
    {
        return salary;
    }
    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    public String getDeg( )
    {
        return deg;
    }
    public void setDeg(String deg)
    {
        this.deg = deg;
    }
    @Override
    public String toString() {
        return "Employee [eid=" + eid + ", ename=" + ename + ", sala
                + salary + ", deg=" + deg + "]";
    }
}
```

在上面的代码中，我们使用@Entity注解，使这个POJO类为实体。

在断续下一个模块前，我们需要对关系实体关联，它记录在 persistence.xml 文件数据库中并创建数据库。打开 MySQL 工作台，然后键入下面的查询。

```
create database jpadb
use jpadb
```

## Persistence.xml

这个模块起着JPA概念至关重要的作用。在这个XML文件中，我们将注册数据库，并指定实体类。

另外，在上述所示的包的层次结构，根据JPA的内容包含在 persistence.xml 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persis
   <persistence-unit name="Eclipselink_JPA"
                     transaction-type="RESOURCE_LOCAL">
    <class>com.yiibai.eclipselink.entity.Employee</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                   value="jdbc:mysql://localhost:3306/jpadb"/>
           <property name="javax.persistence.jdbc.user" value="root
          <property name="javax.persistence.jdbc.password" value="r
          <property name="javax.persistence.jdbc.driver"
                   value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
          <property name="eclipselink.ddl-generation"
                  value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

在上面的XML，**<persistence-unit>**标签定义JPA持久性的特定名称。<class>标记定义了包名的实体类。**<properties>**标签定义的所有属性，而**<property>**标记的每个属性定义，例如数据库注册，URL规范，用户名和密码。这些是EclipseLink属性。此文件将配置数据库。

## 持久化操作

用于与一个数据库进行交互持久性操作，它们加载和存储操作。在一个业务组件，所有的持久化操作属于服务类。

在上面的图示包层次结构，创建一个名为“com.yiibai.eclipselink.service'包，在'src'中（源）封装。所有命名为CreateEmloyee.java，UpdateEmployee.java，FindEmployee.java和DeleteEmployee.java 服务类。来自给定的包下，如下所示：

## 创建**Employee**

下面的代码段说明了如何创建一个名为CreateEmployee.java的一个Employee类。

```
package com.yiibai.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Employee;

public class CreateEmployee
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );

       Employee employee = new Employee( );
       employee.setEid( 1201 );
       employee.setEname( "Gopal" );
       employee.setSalary( 40000 );
       employee.setDeg( "Technical Manager" );
       entitymanager.persist( employee );
       entitymanager.getTransaction( ).commit( );

       entitymanager.close( );
       emfactory.close( );
   }
}
```

在上面的代码中createEntityManagerFactory()通过提供我们在 persistent.xml 文件提供持久化单元相同唯一的名称创建一个持久性单元。 EntityManagerFactory对象将由usingcreateEntityManager()方法创建entitymanger实例。 EntityManager对象创建 entitytransactioninstance 事务管理。通过使用 EntityManager 对象，我们可以持久化实体到数据库中。

编译和执行上述程序后，会从库中的EclipseLink得到通知在Eclipse IDE的控制台面板上。

对于结果，打开 MySQL 工作台，然后输入以下的查询。

```
use jpadb
select * from employee
```

命名为 employee 受影响的数据库表将显示在表格格式如下：

| Eid | Ename | Salary | Deg |
|---|---|---|---|
| 1201 | Gopal | 40000 | Technical Manager |

# 更新Employee

要更新员工的记录，我们需要检索现有记录形成数据库，进行修改，最后提交到数据库中。名为UpdateEmployee.java类如下所示：

```
package com.yiibai.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Employee;

public class UpdateEmployee
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
               entitymanager.getTransaction( ).begin( );
       Employee employee=entitymanager.
               find( Employee.class, 1201 );
       //before update
       System.out.println( employee );
       employee.setSalary( 46000 );
       entitymanager.getTransaction( ).commit( );
        //after update
       System.out.println( employee );
       entitymanager.close();
       emfactory.close();
   }
}
```

编译和执行上述程序后，会从库中的EclipseLink得到通知在Eclipse IDE的控制台面板上。

对于结果，打开MySQL工作台，然后输入以下的查询。

```
use jpadb
select * from employee
```

命名为employee受影响的数据库表将显示在表格格式如下：

| Eid | Ename | Salary | Deg |
|---|---|---|---|
| 1201 | Gopal | 46000 | Technical Manager |

员工的工资为1201将更新为46000。

# 查找Employee

找一个雇员的记录，从数据库中检索现有数据并显示它。在该操作中，EntityTransaction 未在检索的记录中应用。

命名类 FindEmployee.java如下。

```
package com.yiibai.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Employee;

public class FindEmployee
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence
               .createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager();
       Employee employee = entitymanager.
               find( Employee.class, 1201 );

       System.out.println("employee ID = "+employee.getEid( ));
       System.out.println("employee NAME = "+employee.getEname( ))
       System.out.println("employee SALARY = "+employee.getSalary(
       System.out.println("employee DESIGNATION = "+employee.getDeg
   }
}
```

编译并执行上述程序后，从库中的EclipseLink 会得到以下输出在 Eclipse IDE 控制台面板上。

```
employee ID = 1201
employee NAME = Gopal
employee SALARY = 46000.0
employee DESIGNATION = Technical Manager
```

# 删除员工

要删除一个员工的记录，首先我们会发现现有的记录，然后删除它。这里 EntityTransaction 起着重要的作用。

命名 DeleteEmployee.java 类如下：

```
package com.yiibai.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Employee;

public class DeleteEmployee
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );
       Employee employee=entitymanager.
               find( Employee.class, 1201 );
       entitymanager.remove( employee );
       entitymanager.getTransaction( ).commit( );
       entitymanager.close( );
       emfactory.close( );
   }
}
```

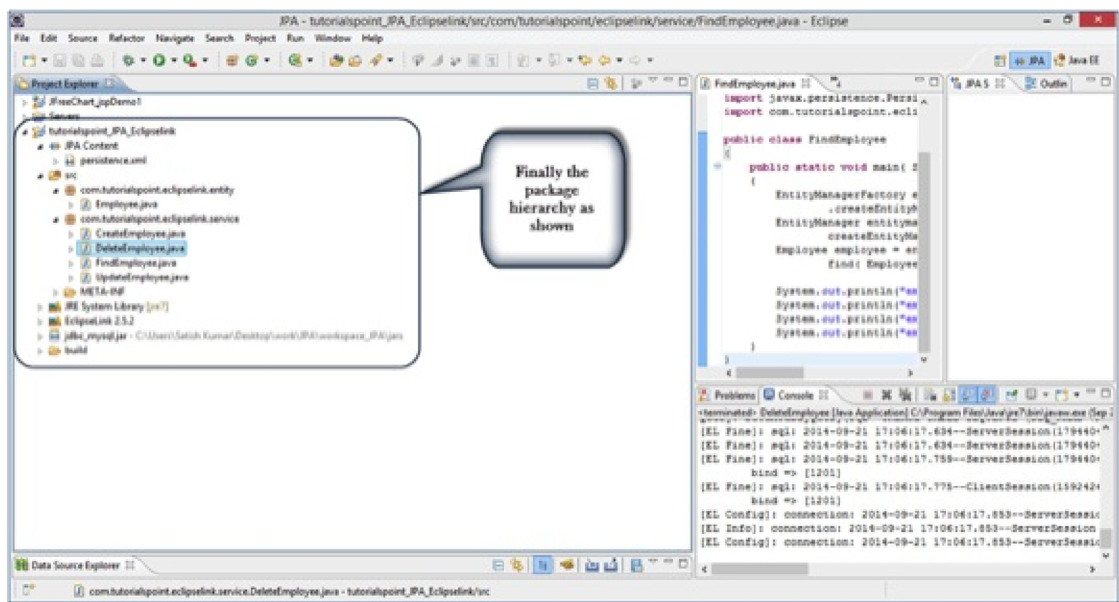
编译和执行上述程序后，会得到Eclipse IDE控制台面板上，从EclipseLink 库中的通知。

对于结果，打开MySQL的工作台，然后键入以下的查询。

```
use jpadb
select * from employee
```

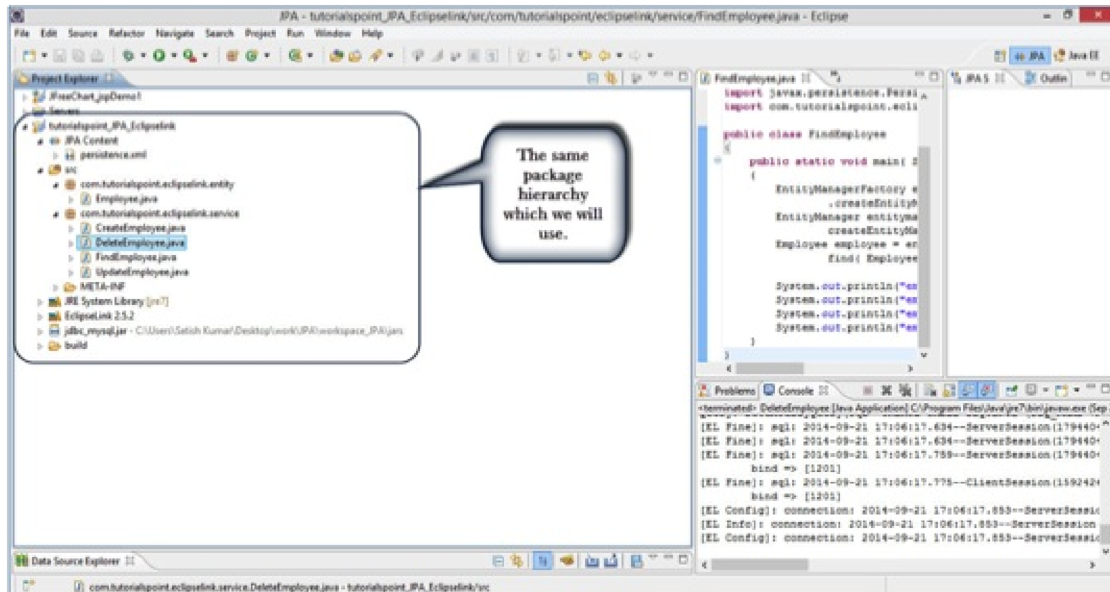
受影响的数据库命名为 employee 的记录为空。

在此示例中所有的模块完成后，将包和文件的层次结构如下所示：


Finally the package hierarchy as shown

# JPA JPQL/持久化查询语言 - JPA教程

本章介绍有关JPQL和它的工作原理与持久性单元。在这一章中，给出的例子遵循相同的包层次结构，和我们在前面的章节中使用一样。




## Java持久化查询语言

JPQL代表Java持久化查询语言。它被用来创建针对实体的查询存储在关系数据库中。 JPQL是基于SQL语法的发展。但它不会直接影响到数据库。

JPQL可以检索使用SELECT子句中的数据，可以使用 UPDATE子句做批量UPDATE和DELETE子句。

## 查询结构

JPQL语法非常类似于SQL语法。SQL的语法是一个优势，因为SQL很简单，被广泛使用。 SQL工作直接针对关系数据库表，记录和字段，而JPQL适用于Java类和实例。

例如，JPQL查询可以检索实体对象，而不是从一个数据库中设置字段结果，作为与SQL。该JPQL查询结构如下。

```
SELECT ... FROM ...
[WHERE ...]
[GROUP BY ... [HAVING ...]]
[ORDER BY ...]
```

JPQL的结构，DELETE和UPDATE查询，如下所示。

```
DELETE FROM ... [WHERE ...]

UPDATE ... SET ... [WHERE ...]
```

## 标量和聚合函数

标量函数返回基于输入值所得的数值。集合函数，通过计算输入值返回的结果值。

我们将使用相同的例子员工管理，在前面的章节。在这里将通过使用JPQL的标量和聚合函数的服务类。

让我们假定 jpadb.employee 表包含下述记录。

| Eid | Ename | Salary | Deg |
|---|---|---|---|
| 1201 | Gopal | 40000 | 技术经理 |
| 1202 | Manisha | 40000 | 接待员 |
| 1203 | Masthanvali | 40000 | 技术作家 |
| 1204 | Satish | 30000 | 技术作家 |
| 1205 | Krishna | 30000 | 技术作家 |
| 1206 | Kiran | 35000 | 接待员 |

创建一个在 com.yiibai.eclipselink.service 包命名为 ScalarandAggregateFunctions.java类如下。

```
package com.yiibai.eclipselink.service;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class ScalarandAggregateFunctions
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager();
       //Scalar function
       Query query = entitymanager.
       createQuery("Select UPPER(e.ename) from Employee e");
       List<String> list=query.getResultList();

       for(String e:list)
       {
           System.out.println("Employee NAME :"+e);
       }
       //Aggregate function
       Query query1 = entitymanager.
               createQuery("Select MAX(e.salary) from Employee e");
       Double result=(Double) query1.getSingleResult();
       System.out.println("Max Employee Salary :"+result);
   }
}
```

编译和执行上面的程序，在Eclipse IDE的控制台面板上会得到以下输出。

```
Employee NAME :GOPAL
Employee NAME :MANISHA
Employee NAME :MASTHANVALI
Employee NAME :SATISH
Employee NAME :KRISHNA
Employee NAME :KIRAN
Max Employee Salary :40000.0
```

# Between, And, Like 关键词

Between, And, 和Like是JPQL的主要关键字。这些关键字在查询子句后使用。

创建一个名为 BetweenAndLikeFunctions.java 类在 com.yiibai.eclipselink.service 包下，如下所示：

```
package com.yiibai.eclipselink.service;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import com.yiibai.eclipselink.entity.Employee;

public class BetweenAndLikeFunctions
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
           createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
           createEntityManager();
       //Between
       Query query = entitymanager.
           createQuery( "Select e " +
               "from Employee e " +
               "where e.salary " +
               "Between 30000 and 40000" )
       List<Employee> list=(List<Employee>)query.getResultList( );

       for( Employee e:list )
       {
           System.out.print("Employee ID :"+e.getEid( ));
           System.out.println("\t Employee salary :"+e.getSalary( 
       }

       //Like
       Query query1 = entitymanager.
           createQuery("Select e " +
               "from Employee e " +
               "where e.ename LIKE 'M%'");
       List<Employee> list1=(List<Employee>)query1.getResultList( 
       for( Employee e:list1 )
       {
           System.out.print("Employee ID :"+e.getEid( ));
           System.out.println("\t Employee name :"+e.getEname( ));
       }
   }
}
```

编译并执行上述程序后，将在Eclipse IDE的控制台面板下面输出以下内容。

```
Employee ID :1201    Employee salary :40000.0
Employee ID :1202    Employee salary :40000.0
Employee ID :1203    Employee salary :40000.0
Employee ID :1204    Employee salary :30000.0
Employee ID :1205    Employee salary :30000.0
Employee ID :1206    Employee salary :35000.0

Employee ID :1202    Employee name :Manisha
Employee ID :1203    Employee name :Masthanvali
```

# 排序

要排序JPQL中的记录，我们使用ORDER BY子句。这一个子句的使用类似于SQL中的用法，但它涉及的实体。下面的示例演示了如何使用ORDER BY子句。

在com.yiibai.eclipselink.service包中创建类 Ordering.java 如下：

```
package com.yiibai.eclipselink.service;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import com.yiibai.eclipselink.entity.Employee;

public class Ordering
{
   public static void main( String[ ] args )
   {
      EntityManagerFactory emfactory = Persistence.
         createEntityManagerFactory( "Eclipselink_JPA" );
      EntityManager entitymanager = emfactory.
         createEntityManager();
      //Between
      Query query = entitymanager.
         createQuery( "Select e " +
            "from Employee e " +
            "ORDER BY e.ename ASC" );
      List<Employee> list=(List<Employee>)query.getResultList( );

      for( Employee e:list )
      {
         System.out.print("Employee ID :"+e.getEid( ));
         System.out.println("\t Employee Name :"+e.getEname( ));
      }
   }
}
```

编译和执行上面的程序，在Eclipse IDE的控制台面板会产生下面的输出。

```
Employee ID :1201    Employee Name :Gopal
Employee ID :1206    Employee Name :Kiran
Employee ID :1205    Employee Name :Krishna
Employee ID :1202    Employee Name :Manisha
Employee ID :1203    Employee Name :Masthanvali
Employee ID :1204    Employee Name :Satish
```

## 命名查询

@NamedQuery注解被定义为一个预定义的查询字符串，它是不可改变的查询。相反，动态查询，命名查询可以通过POJO分离JPQL查询字符串提高代码的组织。它也传送的查询参数，而不是动态地嵌入文本到查询字符串，并因此产生更高效的查询。

首先，@NamedQuery注解添加到com.yiibai.eclipselink.entity包中的 Employee实体，类名为Employee.java下，如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@Table
@NamedQuery(query = "Select e from Employee e where e.eid = :id",
   name = "find employee by id")
public class Employee
{
   @Id
   @GeneratedValue(strategy= GenerationType.AUTO)
   private int eid;
   private String ename;
   private double salary;
   private String deg;
   public Employee(int eid, String ename, double salary, String deg
   {
       super( );
       this.eid = eid;
       this.ename = ename;
       this.salary = salary;
       this.deg = deg;
   }
   public Employee( )
```

```
    {
        super();
    }

    public int getEid( )
    {
        return eid;
    }
    public void setEid(int eid)
    {
        this.eid = eid;
    }

    public String getEname( )
    {
        return ename;
    }
    public void setEname(String ename)
    {
        this.ename = ename;
    }

    public double getSalary( )
    {
        return salary;
    }
    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    public String getDeg( )
    {
        return deg;
    }
    public void setDeg(String deg)
    {
        this.deg = deg;
    }
    @Override
    public String toString() {
        return "Employee [eid=" + eid + ", ename=" + ename + ", sala
                + salary + ", deg=" + deg + "]";
    }
}
```

创建一个名为com.yiibai.eclipselink.service包下的NamedQueries.java类，如下所示：

```
package com.yiibai.eclipselink.service;

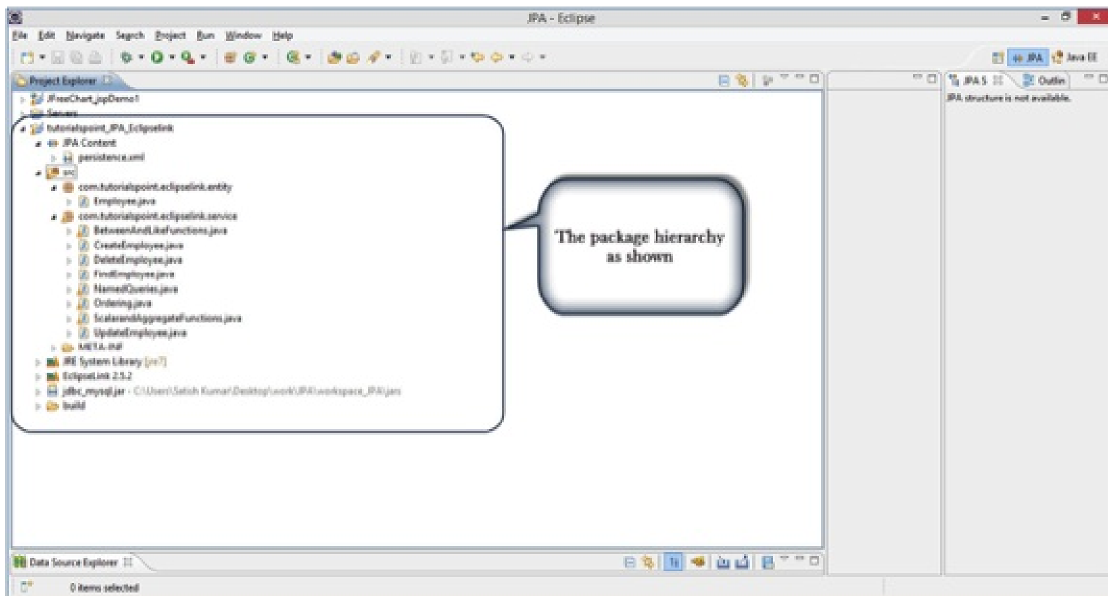
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import com.yiibai.eclipselink.entity.Employee;

public class NamedQueries
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
           createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
           createEntityManager();
       Query query = entitymanager.createNamedQuery(
           "find employee by id");
       query.setParameter("id", 1204);
       List<Employee> list = query.getResultList( );
       for( Employee e:list )
       {
           System.out.print("Employee ID :"+e.getEid( ));
           System.out.println("\t Employee Name :"+e.getEname( ));
       }
   }
}
```

经过编译和执行上面的程序，在Eclipse IDE的控制台面板上会得到下面的输出。

```
Employee ID :1204    Employee Name :Satish
```

加入上述所有类后，包层次结构如下所示：




# 急切和延迟加载

JPA中最重要的概念是为了使数据库的副本在高速缓冲存储器中。虽然有一个数据库事务，但JPA首先创建一个重复的数据集，只有当它使用实体管理提交，所做的更改影响到数据库中。

从数据库中获取记录有两种方式。

## 预先抓取

在预先抓取，相关的子对象获取一个特定的记录自动上传。

## 延迟加载

在延迟装载，涉及的对象不会自动上传，除非你特别要求他们。首先，它检查相关对象和通知可用性。以后，如果调用任何实体的getter方法，那么它获取的所有记录。

延迟装载可能在第一次尝试获取记录。这样一来，在整个记录的副本已经被存储在高速缓冲存储器中。性能方面，延迟装载最好。

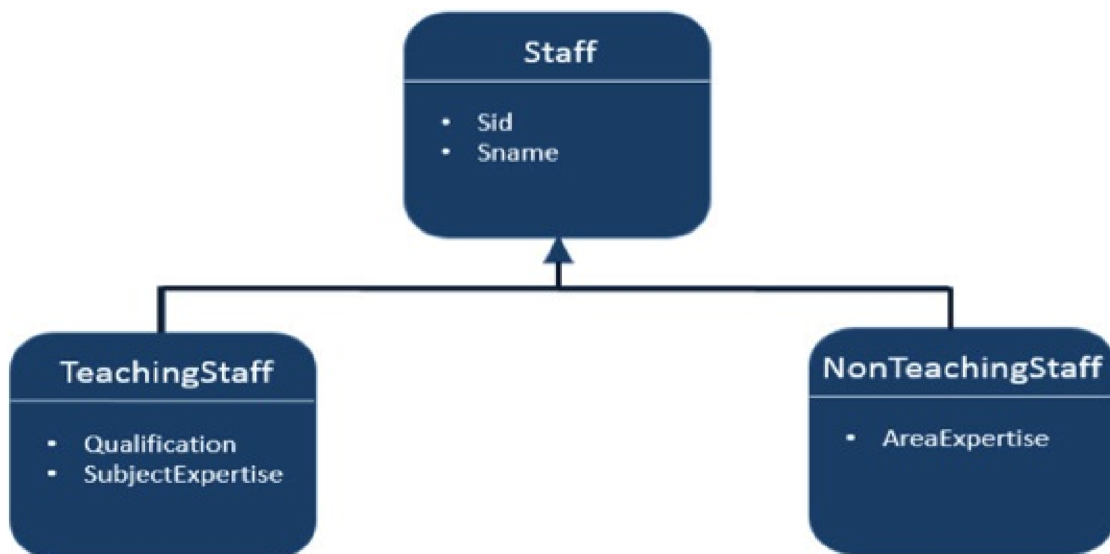
# JPA高级映射 - JPA教程

JPA是一种发布使用Java规范的库。因此，它支持所有的面向对象的概念实体，持久性。到现在为止，我们已经完成了对象关系映射的基本知识。本章将完成对象和关系单位之间的高级映射。

## 继承策略

继承是任何面向对象语言的核心理念，因此我们可以用实体之间的继承关系和策略。JPA支持三种类型的继承策略：SINGLE_TABLE，JOINED_TABLE和TABLE_PER_CONCRETE_CLASS。

让我们考虑一个例子。下图显示了三个等级，即Staff, TeachingStaff, and NonTeachingStaff和他们之间的关系。




在上面的图中，员工是一个实体，而TeachingStaff和NonTeachingStaff是工作人员的子实体。在这里，我们将使用上面的例子来演示继承的全部三个策略。

## 单一表策略

单表策略采取所有类的字段(包括超级亚类)，并将它们映射成称为SINGLE_TABLE策略一个表。这里的鉴别值起着区分在一个表中三个实体的值的关键作用。

让我们考虑上面的例子。TeachingStaff和NonTeachingStaff是Staff的子类。按照继承的概念，一个子类继承其超类中的属性。因此sid和sname属于TeachingStaff和NonTeachingStaff 字段。创建JPA项目。在这个项目中的所有模块，如下所示：

### 创建实体

创建一个名为“src”（源）在“com.yiibai.eclipselink.entity'包下。创建一个名为Staff.java新的Java类。工作人员实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import java.io.Serializable;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;
@Entity
@Table
@Inheritance( strategy = InheritanceType.SINGLE_TABLE )
@DiscriminatorColumn( name="type" )
public class Staff implements Serializable
{
   @Id
   @GeneratedValue( strategy = GenerationType.AUTO )
   private int sid;
   private String sname;
   public Staff( int sid, String sname )
   {
       super( );
       this.sid = sid;
       this.sname = sname;
   }
   public Staff( )
   {
       super( );
   }
   public int getSid( )
   {
       return sid;
   }
   public void setSid( int sid )
   {
       this.sid = sid;
   }
   public String getSname( )
   {
       return sname;
   }
   public void setSname( String sname )
   {
       this.sname = sname;
   }
}
```

在上面的代码@DescriminatorColumn指定字段名称（类型）和它的值显示剩余（Teaching和NonTeachingStaff）字段。

创建Staff类的一个子类叫TeachingStaff.java在com.yiibai.eclipselink.entity包下。TeachingStaff 实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue( value="TS" )
public class TeachingStaff extends Staff
{
   private String qualification;
   private String subjectexpertise;

   public TeachingStaff( int sid, String sname,
           String qualification,String subjectexpertise )
   {
       super( sid, sname );
       this.qualification = qualification;
       this.subjectexpertise = subjectexpertise;
   }

   public TeachingStaff( )
   {
       super( );
   }

   public String getQualification( )
   {
       return qualification;
   }

   public void setQualification( String qualification )
   {
       this.qualification = qualification;
   }

   public String getSubjectexpertise( )
   {
       return subjectexpertise;
   }

   public void setSubjectexpertise( String subjectexpertise )
   {
       this.subjectexpertise = subjectexpertise;
   }
}
```

创建Staff 的一个子类（类）类叫NonTeachingStaff.java 在 com.yiibai.eclipselink.entity 包下。NonTeachingStaff实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue( value = "NS" )
public class NonTeachingStaff extends Staff
{
   private String areaexpertise;

   public NonTeachingStaff( int sid, String sname,
           String areaexpertise )
   {
       super( sid, sname );
       this.areaexpertise = areaexpertise;
   }

   public NonTeachingStaff( )
   {
       super( );
   }

   public String getAreaexpertise( )
   {
       return areaexpertise;
   }

   public void setAreaexpertise( String areaexpertise )
   {
       this.areaexpertise = areaexpertise;
   }
}
```

## Persistence.xml

persistence.xml中包含数据库的配置信息和实体类的注册信息。 XML文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persis
   <persistence-unit name="Eclipselink_JPA"
                        transaction-type="RESOURCE_LOCAL">
       <class>com.yiibai.eclipselink.entity.Staff</class>
       <class>com.yiibai.eclipselink.entity.NonTeachingStaff</class
       <class>com.yiibai.eclipselink.entity.TeachingStaff</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                            value="jdbc:mysql://localhost:3306/jpad
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                            value="root"/>
           <property name="javax.persistence.jdbc.driver"
                            value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
           <property name="eclipselink.ddl-generation"
                            value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

## 服务类

服务类业务组件的实现部分。创建com.yiibai.eclipselink.service“包在'src”下。

创建一个给定的包下名为SaveClient.java用来存储Staff，TeachingStaff和NonTeachingStaff类字段类。SaveClient类如下所示：

```
package com.yiibai.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.NonTeachingStaff;
import com.yiibai.eclipselink.entity.TeachingStaff;

public class SaveClient
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );

       //Teaching staff entity
       TeachingStaff ts1=new TeachingStaff(
               1,"Gopal","MSc MEd","Maths");
       TeachingStaff ts2=new TeachingStaff(
               2, "Manisha", "BSc BEd", "English");
       //Non-Teaching Staff entity
       NonTeachingStaff nts1=new NonTeachingStaff(
               3, "Satish", "Accounts");
       NonTeachingStaff nts2=new NonTeachingStaff(
               4, "Krishna", "Office Admin");

       //storing all entities
       entitymanager.persist(ts1);
       entitymanager.persist(ts2);
       entitymanager.persist(nts1);
       entitymanager.persist(nts2);
       entitymanager.getTransaction().commit();
       entitymanager.close();
       emfactory.close();
   }
}
```

编译并执行上述程序后，Eclipse IDE的控制台面板上会得到通知。检查MySQL工作台的输出。以表格格式的输出如下所示：

| Sid | Type | Sname | Areaexpertise | Qualification | Subjectexpertise |
|---|---|---|---|---|---|
| 1 | TS | Gopal | MSC MED | Maths | |
| 2 | TS | Manisha | BSC BED | English | |
| 3 | NS | Satish | Accounts | | |
| 4 | NS | Krishna | Office Admin | | |

最后，会得到一个包含所有三个类字段鉴别列命名类型（字段）一个表。

## 注册策略表

连接表的策略是共享引用的列包含唯一值的加入表并进行便捷事务。让我们考虑与上述相同的例子。

创建JPA项目。所有工程的模块如下所示。

### 创建实体

在“src”下创建一个名为 com.yiibai.eclipselink.entity' 的包。创建一个名为Staff.java新的Java类。Staff 实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table
@Inheritance( strategy = InheritanceType.JOINED )
public class Staff implements Serializable
{
   @Id
   @GeneratedValue( strategy = GenerationType.AUTO )
   private int sid;
   private String sname;
   public Staff( int sid, String sname )
   {
       super( );
       this.sid = sid;
       this.sname = sname;
   }
   public Staff( )
   {
       super( );
   }
   public int getSid( )
   {
       return sid;
   }
   public void setSid( int sid )
   {
       this.sid = sid;
   }
   public String getSname( )
   {
       return sname;
   }
   public void setSname( String sname )
   {
       this.sname = sname;
   }
}
```

创建Staff类的一个子类叫TeachingStaff.java在com.yiibai.eclipselink.entity包下。TeachingStaff 实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@PrimaryKeyJoinColumn(referencedColumnName="sid")
public class TeachingStaff extends Staff
{
   private String qualification;
   private String subjectexpertise;

   public TeachingStaff( int sid, String sname,
           String qualification,String subjectexpertise )
   {
       super( sid, sname );
       this.qualification = qualification;
       this.subjectexpertise = subjectexpertise;
   }

   public TeachingStaff( )
   {
       super( );

   }

   public String getQualification( )
   {
       return qualification;
   }

   public void setQualification( String qualification )
   {
       this.qualification = qualification;
   }

   public String getSubjectexpertise( )
   {
       return subjectexpertise;
   }

   public void setSubjectexpertise( String subjectexpertise )
   {
       this.subjectexpertise = subjectexpertise;
   }
}
```

创建Staff的一个子类（类）类叫NonTeachingStaff.java在com.yiibai.eclipselink.entity包下。NonTeachingStaff实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@PrimaryKeyJoinColumn(referencedColumnName="sid")
public class NonTeachingStaff extends Staff
{
   private String areaexpertise;

   public NonTeachingStaff( int sid, String sname,
           String areaexpertise )
   {
       super( sid, sname );
       this.areaexpertise = areaexpertise;
   }

   public NonTeachingStaff( )
   {
       super( );
   }

   public String getAreaexpertise( )
   {
       return areaexpertise;
   }

   public void setAreaexpertise( String areaexpertise )
   {
       this.areaexpertise = areaexpertise;
   }
}
```

## Persistence.xml

persistence.xml文件包含数据库的配置信息和实体类的注册信息。 XML文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persis
   <persistence-unit name="Eclipselink_JPA"
                        transaction-type="RESOURCE_LOCAL">
   <class>com.yiibai.eclipselink.entity.Staff</class>
   <class>com.yiibai.eclipselink.entity.NonTeachingStaff</class>
   <class>com.yiibai.eclipselink.entity.TeachingStaff</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                            value="jdbc:mysql://localhost:3306/jpac
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                            value="root"/>
           <property name="javax.persistence.jdbc.driver"
                            value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
           <property name="eclipselink.ddl-generation"
                            value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

## 服务类

服务类业务组件的实现部分。在src下创建一个包com.yiibai.eclipselink.service“。

创建一个名为SaveClient.java给定的包来存储Staff， TeachingStaff, NonTeachingStaff类来存储字段。然后SaveClient类如下所示：

```
package com.yiibai.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.NonTeachingStaff;
import com.yiibai.eclipselink.entity.TeachingStaff;

public class SaveClient
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );

       //Teaching staff entity
       TeachingStaff ts1=new TeachingStaff(
               1,"Gopal","MSc MEd","Maths");
       TeachingStaff ts2=new TeachingStaff(
               2, "Manisha", "BSc BEd", "English");
       //Non-Teaching Staff entity
       NonTeachingStaff nts1=new NonTeachingStaff(
               3, "Satish", "Accounts");
       NonTeachingStaff nts2=new NonTeachingStaff(
       4, "Krishna", "Office Admin");

       //storing all entities
       entitymanager.persist(ts1);
       entitymanager.persist(ts2);
       entitymanager.persist(nts1);
       entitymanager.persist(nts2);

       entitymanager.getTransaction().commit();
       entitymanager.close();
       emfactory.close();
   }
}
```

编译和执行上述程序后，在Eclipse IDE的控制台面板得到通知。对于输出，检查MySQL工作台。

在这里，将创建三个表和工作人员表的结果显示在表格格式。

| Sid | Dtype | Sname |
|---|---|---|
| 1 | TeachingStaff | Gopal |
| 2 | TeachingStaff | Manisha |
| 3 | NonTeachingStaff | Satish |
| 4 | NonTeachingStaff | Krishna |

TeachingStaff表的结果显示如下：

| Sid | Qualification | Subjectexpertise |
|---|---|---|
| 1 | MSC MED | Maths |
| 2 | BSC BED | English |

在上表中的sid是外键（参考字段工作人员表单表）NonTeachingStaff 表的结果显示如下：

| Sid | Areaexpertise |
|---|---|
| 3 | Accounts |
| 4 | Office Admin |

最后，使用各自字段中创建三个表和SID字段由所有三个表共享。在员工表中，SID是主键。在剩下的两个表（TeachingStaff和NonTeachingStaff），SID是外键。

# 每个类表策略

表每个类策略是创建一个表中为每个子实体。 Staff表将被创建，但它会包含空值。Staff 表的字段值必须同时由TeachingStaff和NonTeachingStaff表共享。

让我们考虑与上述相同的例子。

## 创建实体

在src下创建一个名为“com.yiibai.eclipselink.entity'的包。创建一个名为Staff.java为一个新的Java类。Staff实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table
@Inheritance( strategy = InheritanceType.TABLE_PER_CLASS )
public class Staff implements Serializable
{
   @Id
   @GeneratedValue( strategy = GenerationType.AUTO )
   private int sid;
   private String sname;
   public Staff( int sid, String sname )
   {
       super( );
       this.sid = sid;
       this.sname = sname;
   }
   public Staff( )
   {
       super( );
   }
   public int getSid( )
   {
       return sid;
   }
   public void setSid( int sid )
   {
       this.sid = sid;
   }
   public String getSname( )
   {
       return sname;
   }
   public void setSname( String sname )
   {
       this.sname = sname;
   }
}
```

创建Staff类的一个子类叫TeachingStaff.java 在com.yiibai.eclipselink.entity包下。该TeachingStaff实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
public class TeachingStaff extends Staff
{
   private String qualification;
   private String subjectexpertise;

   public TeachingStaff( int sid, String sname,
           String qualification,String subjectexpertise )
   {
       super( sid, sname );
       this.qualification = qualification;
       this.subjectexpertise = subjectexpertise;
   }

   public TeachingStaff( )
   {
       super( );

   }

   public String getQualification( )
   {
       return qualification;
   }
   public void setQualification( String qualification )
   {
       this.qualification = qualification;
   }

   public String getSubjectexpertise( )
   {
       return subjectexpertise;
   }

   public void setSubjectexpertise( String subjectexpertise )
   {
       this.subjectexpertise = subjectexpertise;
   }
}
```

创建Staff 的一个子类（类）类叫NonTeachingStaff.java在com.yiibai.eclipselink.entity包下。NonTeachingStaff实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
public class NonTeachingStaff extends Staff
{
   private String areaexpertise;

   public NonTeachingStaff( int sid, String sname,
           String areaexpertise )
           {
       super( sid, sname );
       this.areaexpertise = areaexpertise;
   }

   public NonTeachingStaff( )
   {
       super( );
   }

   public String getAreaexpertise( )
   {
       return areaexpertise;
   }

   public void setAreaexpertise( String areaexpertise )
   {
       this.areaexpertise = areaexpertise;
   }
}
```

## Persistence.xml

persistence.xml文件中包含的实体类的数据库和注册信息的配置信息。 XML文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persis
   <persistence-unit name="Eclipselink_JPA"
                     transaction-type="RESOURCE_LOCAL">
   <class>com.yiibai.eclipselink.entity.Staff</class>
   <class>com.yiibai.eclipselink.entity.NonTeachingStaff</class>
   <class>com.yiibai.eclipselink.entity.TeachingStaff</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                            value="jdbc:mysql://localhost:3306/jpa
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                            value="root"/>
           <property name="javax.persistence.jdbc.driver"
                            value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
           <property name="eclipselink.ddl-generation"
                            value="create-tables"/>
           </properties>
   </persistence-unit>
</persistence>
```

## 服务类

服务类业务组件的实现部分。在src下创建一个包com.yiibai.eclipselink.service“。

创建一个给定的包下名为SaveClient.java用来存储Staff，TeachingStaff和NonTeachingStaff 类字段类。该SaveClient类如下所示：

```
package com.yiibai.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.NonTeachingStaff;
import com.yiibai.eclipselink.entity.TeachingStaff;
public class SaveClient
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );

       //Teaching staff entity
       TeachingStaff ts1=new TeachingStaff(
               1,"Gopal","MSc MEd","Maths");
       TeachingStaff ts2=new TeachingStaff(
               2, "Manisha", "BSc BEd", "English");
       //Non-Teaching Staff entity
       NonTeachingStaff nts1=new NonTeachingStaff(
               3, "Satish", "Accounts");
       NonTeachingStaff nts2=new NonTeachingStaff(
               4, "Krishna", "Office Admin");

       //storing all entities
       entitymanager.persist(ts1);
       entitymanager.persist(ts2);
       entitymanager.persist(nts1);
       entitymanager.persist(nts2);

       entitymanager.getTransaction().commit();
       entitymanager.close();
       emfactory.close();
   }
}
```

编译并执行上述程序后，Eclipse IDE的控制台面板上会得到通知。对于输出，检查MySQL的工作台。

这里创建了三个表并且Staff表的记录为空。

TeachingStaff的结果显示如下：

| Sid | Qualification | Sname | Subjectexpertise |
|---|---|---|---|
| 1 | MSC MED | Gopal | Maths |
| 2 | BSC BED | Manisha | English |

上表TeachingStaff包含Staff和TeachingStaff实体字段。

NonTeachingStaff的结果显示如下：

| Sid | Areaexpertise | Sname |
|---|---|---|
| 3 | Accounts | Satish |
| 4 | Office Admin | Krishna |

上表NonTeachingStaff包含Staff和NonTeachingStaff实体字段。

# JPA实体关系 - JPA教程

本章将指导完成学习实体间的关系。一般的关系数据库中的表之间的更有效。这里的实体类都被视为关系表(JPA的概念)，因此是实体类之间的关系如下：

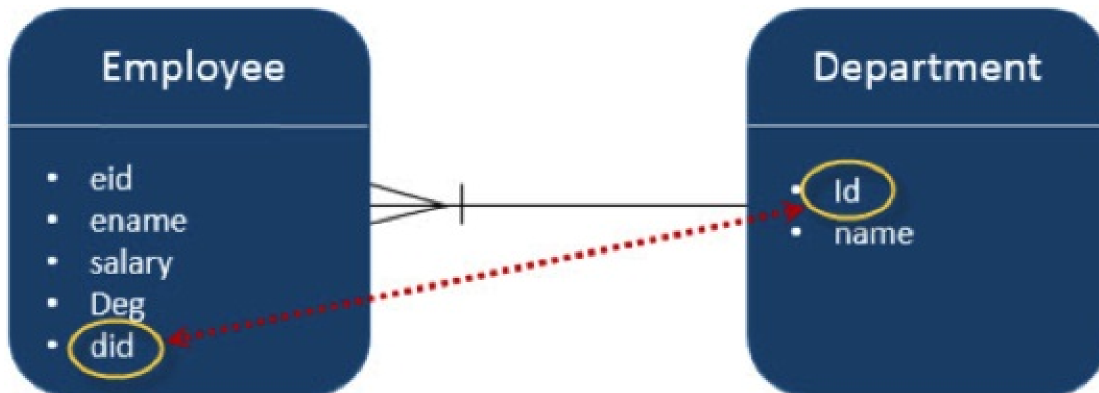
- @ManyToOne 关系
- @OneToMany 关系
- @OneToOne 关系
- @ManyToMany 关系

## @ManyToOne 关系

多对一实体之间存在的关系，其中一个实体（列或组列）的引用与另一个实体（列或组列）包含唯一值。在关系数据库中，这些关系是通过使用表之间的外键/主键应用。

让我们考虑Employee和Department 实体之间的关系的一个例子。在单向方式，即从员工到部门，多到一的关系是可行的。这意味着员工每个记录包含一个部门ID，它应该是在部门表的主键。在这里，在Employee表，Department 的ID是外键。

下图显示了两个表之间的多对一关系。




在Eclipse中创建一个IDE JPA项目命名为JPA_Eclipselink_MTO。这个项目的所有模块下面讨论。

### 创建实体

遵循用于创建实体上面给出的图，在src下创建“com.tutorialspoin.eclipselink.entity'包。创建一个名为Department.java的类。类系实体如下：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Department
{
    @Id
    @GeneratedValue( strategy=GenerationType.AUTO )
    private int id;
    private String name;

    public int getId()
    {
       return id;
    }

    public void setId(int id)
    {
       this.id = id;
    }

    public String getName( )
    {
       return name;
    }

    public void setName( String deptName )
    {
       this.name = deptName;
    }
}
```

建立在这种关系中的第二个实体 - 名为Employee.java 在“com.yiibai.eclipselink.entity'包下的Employee实体类。 Employee实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
public class Employee
```

```
{
   @Id
   @GeneratedValue( strategy= GenerationType.AUTO )
   private int eid;
   private String ename;
   private double salary;
   private String deg;
   @ManyToOne
   private Department department;

   public Employee(int eid,
           String ename, double salary, String deg)
   {
       super( );
       this.eid = eid;
       this.ename = ename;
       this.salary = salary;
       this.deg = deg;
   }

   public Employee( )
   {
       super();
   }

   public int getEid( )
   {
       return eid;
   }
   public void setEid(int eid)
   {
       this.eid = eid;
   }

   public String getEname( )
   {
       return ename;
   }
   public void setEname(String ename)
   {
       this.ename = ename;
   }

   public double getSalary( )
   {
       return salary;
   }
   public void setSalary(double salary)
   {
       this.salary = salary;
   }

   public String getDeg( )
```

```
    {
        return deg;
    }
    public void setDeg(String deg)
    {
        this.deg = deg;
    }

    public Department getDepartment() {
        return department;
    }

    public void setDepartment(Department department) {
        this.department = department;
    }
}
```

## Persistence.xml

persistence.xml文件需要配置数据库和实体类的注册。

Persitence.xml由Eclipse IDE在创建时由JPA项目创建。详细配置信息是用户的规范。 persistence.xml 文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persist
             http://java.sun.com/xml/ns/persistence/persistence_2_(
   <persistence-unit name="Eclipselink_JPA"
                          transaction-type="RESOURCE_LOCAL">
   <class>com.yiibai.eclipselink.entity.Employee</class>
   <class>com.yiibai.eclipselink.entity.Department</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                     value="jdbc:mysql://localhost:3306/jpadb"/>
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                            value="root"/>
           <property name="javax.persistence.jdbc.driver"
                     value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
           <property name="eclipselink.ddl-generation"
                     value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

## 服务类

该模块包含服务类，它实现了使用属性初始化关系的一部分。在src下创建一个名为“com.yiibai.eclipselink.service'包。创建一个名为ManyToOne.java的DAO类。在DAO类如下所示：

```
package com.yiibaieclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Department;
import com.yiibai.eclipselink.entity.Employee;

public class ManyToOne
{
   public static void main( String[ ] args )
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );

       //Create Department Entity
       Department department = new Department();
       department.setName("Development");
       //Store Department
       entitymanager.persist(department);

       //Create Employee1 Entity
       Employee employee1 = new Employee();
       employee1.setEname("Satish");
       employee1.setSalary(45000.0);
       employee1.setDeg("Technical Writer");
       employee1.setDepartment(department);

       //Create Employee2 Entity
       Employee employee2 = new Employee();
       employee2.setEname("Krishna");
       employee2.setSalary(45000.0);
       employee2.setDeg("Technical Writer");
       employee2.setDepartment(department);

       //Create Employee3 Entity
       Employee employee3 = new Employee();
       employee3.setEname("Masthanvali");
       employee3.setSalary(50000.0);
       employee3.setDeg("Technical Writer");
       employee3.setDepartment(department);

       //Store Employees
```

```
        entitymanager.persist(employee1);
        entitymanager.persist(employee2);
        entitymanager.persist(employee3);

        entitymanager.getTransaction().commit();
        entitymanager.close();
        emfactory.close();
    }
}
```

编译并执行上述程序后，Eclipse IDE控制台面板上会显示的通知。对于输出，检查MySQL工作台。在此示例中，将创建两个表。

通过在MySQL管理界面中执行下面的查询，Department表的结果显示如下：

```
Select * from department
```

| ID | Name |
|---|---|
| 101 | Development |

通过在MySQL界面执行下面的查询，Employee表显示如下的结果。

```
Select * from employee
```

| Eid | Deg | Ename | Salary | Department_Id |
|---|---|---|---|---|
| 102 | Technical Writer | Satish | 45000 | 101 |
| 103 | Technical Writer | Krishna | 45000 | 101 |
| 104 | Technical Writer | Masthanwali | 50000 | 101 |

在上表中Deparment_Id是Department表的外键（参考字段）。

# @OneToMany 关系

在这种关系中，一个实体中的每一行被引用到其它实体的许多子记录。最重要的是，子记录不能有多个父。在表A和表B之间的1对多的关系，在表A中的每一行都可以在表B中被链接到一个或多个行

让我们考虑上面的例子。假设Employee 和Department在上述例子中的表连接在一个相反的单向方式，那么关系成为一个一对多的关系。在Eclipse中创建一个IDE JPA项目命名JPA_Eclipselink_OTM。这个项目的所有模块下面讨论。

## 创建实体

遵循用于创建实体上面给出的图，在src下创建“com.tutorialspoin.eclipselink.entity'包。创建一个名为Department.java的类。类系实体如下：

```
package com.yiibai.eclipselink.entity;

import java.util.List;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class Department
{
    @Id
    @GeneratedValue( strategy=GenerationType.AUTO )
    private int id;
    private String name;

    @OneToMany( targetEntity=Employee.class )
    private List employeelist;

    public int getId()
    {
       return id;
    }

    public void setId(int id)
    {
       this.id = id;
    }

    public String getName( )
    {
       return name;
    }

    public void setName( String deptName )
    {
       this.name = deptName;
    }

    public List getEmployeelist()
    {
   return employeelist;
    }

    public void setEmployeelist(List employeelist)
    {
   this.employeelist = employeelist;
    }
}
```

创建第二个实体类关系-Employee实体类，命名为Employee.java 在“com.yiibai.eclipselink.entity'包下。 Employee实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Employee
{
   @Id
   @GeneratedValue( strategy= GenerationType.AUTO )
   private int eid;
   private String ename;
   private double salary;
   private String deg;

   public Employee(int eid,
           String ename, double salary, String deg)
   {
       super( );
       this.eid = eid;
       this.ename = ename;
       this.salary = salary;
       this.deg = deg;
   }

   public Employee( )
   {
       super();
   }

   public int getEid( )
   {
       return eid;
   }
   public void setEid(int eid)
   {
       this.eid = eid;
   }

   public String getEname( )
   {
       return ename;
   }
   public void setEname(String ename)
   {
       this.ename = ename;
   }
```

```
    public double getSalary( )
    {
        return salary;
    }
    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    public String getDeg( )
    {
        return deg;
    }
    public void setDeg(String deg)
    {
        this.deg = deg;
    }
}
```

## Persistence.xml

persistence.xml文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persist
             http://java.sun.com/xml/ns/persistence/persistence_2_(
   <persistence-unit name="Eclipselink_JPA"
                         transaction-type="RESOURCE_LOCAL">
   <class>com.yiibai.eclipselink.entity.Employee</class>
   <class>com.yiibai.eclipselink.entity.Department</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                     value="jdbc:mysql://localhost:3306/jpadb"/>
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                            value="root"/>
           <property name="javax.persistence.jdbc.driver"
                     value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
           <property name="eclipselink.ddl-generation"
                     value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

## 服务类

该模块包含服务类，它实现了使用属性初始化关系的一部分。在src下创建一个名为“com.yiibai.eclipselink.service'包。并在包下创建一个名为OneToMany.java的DAO类。在DAO类如下所示：

```
package com.yiibaieclipselink.service;

import java.util.List;
import java.util.ArrayList;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Department;
import com.yiibai.eclipselink.entity.Employee;

public class OneToMany
{
   public static void main(String[] args)
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );

       //Create Employee1 Entity
       Employee employee1 = new Employee();
       employee1.setEname("Satish");
       employee1.setSalary(45000.0);
       employee1.setDeg("Technical Writer");

       //Create Employee2 Entity
       Employee employee2 = new Employee();
       employee2.setEname("Krishna");
       employee2.setSalary(45000.0);
       employee2.setDeg("Technical Writer");

       //Create Employee3 Entity
       Employee employee3 = new Employee();
       employee3.setEname("Masthanvali");
       employee3.setSalary(50000.0);
       employee3.setDeg("Technical Writer");

       //Store Employee
       entitymanager.persist(employee1);
       entitymanager.persist(employee2);
       entitymanager.persist(employee3);

       //Create Employeelist
       List<Employee> emplist = new ArrayList();
       emplist.add(employee1);
```

```
        emplist.add(employee2);
        emplist.add(employee3);

        //Create Department Entity
        Department department= new Department();
        department.setName("Development");
        department.setEmployeelist(emplist);

        //Store Department
        entitymanager.persist(department);

        entitymanager.getTransaction().commit();
        entitymanager.close();
        emfactory.close();
    }
}
```

编译和执行上述程序后，在Eclipse IDE的控制台面板中得到通知。对于输出检查MySQL工作台如下。

在这个项目中创建三个表。通过在MySQL界面查询，department_employee表的结果显示如下：

```
Select * from department_Id;
```

| Department_ID | Employee_Eid |
|---|---|
| 254 | 251 |
| 254 | 252 |
| 254 | 253 |

在上表中，deparment_id和employee_id 是部门和员工表的外键（参考字段）。

通过在MySQL界面下面查询，department表的结果将显示如下表格格式。

```
Select * from department;
```

| ID | Name |
|---|---|
| 254 | Development |

通过在MySQL界面下面查询，employee表的结果显示如下：

```
Select * from employee;
```

| Eid | Deg | Ename | Salary |
|---|---|---|---|
| 251 | Technical Writer | Satish | 45000 |
| 252 | Technical Writer | Krishna | 45000 |
| 253 | Technical Writer | Masthanwali | 50000 |

# @OneToOne 关系

在一对一关系，一个项可以链接到只能另一个项。这意味着一个实体中的每一行被称为一个且仅一个行对另一个实体。

让我们考虑上面的例子。Employee和Department在反向单向的方式，关系是一对一的关系。这意味着每个员工只能属于一个部门。 在Eclipse中创建一个IDE JPA项目命名JPA_Eclipselink_OTO。这个项目的所有模块下面讨论。

## 创建实体

遵循用于创建实体上面给出的图。在src下创建一个名为“com.tutorialspoin.eclipselink.entity'的包。在这个包下创建类名为：Department.java。类系实体被示为如下：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Department
{
    @Id
    @GeneratedValue( strategy=GenerationType.AUTO )
    private int id;
    private String name;

    public int getId()
    {
       return id;
    }

    public void setId(int id)
    {
       this.id = id;
    }

    public String getName( )
    {
       return name;
    }

    public void setName( String deptName )
    {
       this.name = deptName;
    }
}
```

创建第二个实体类关系-Employee实体类，命名为Employee.java
在“com.yiibai.eclipselink.entity'包下。 Employee实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

@Entity
public class Employee
{
```

```
    @Id
    @GeneratedValue( strategy= GenerationType.AUTO )
    private int eid;
    private String ename;
    private double salary;
    private String deg;

    @OneToOne
    private Department department;

    public Employee(int eid,
            String ename, double salary, String deg)
    {
        super( );
        this.eid = eid;
        this.ename = ename;
        this.salary = salary;
        this.deg = deg;
    }

    public Employee( )
    {
        super();
    }

    public int getEid( )
    {
        return eid;
    }
    public void setEid(int eid)
    {
        this.eid = eid;
    }

    public String getEname( )
    {
        return ename;
    }
    public void setEname(String ename)
    {
        this.ename = ename;
    }

    public double getSalary( )
    {
        return salary;
    }
    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    public String getDeg( )
```

```
    {
        return deg;
    }
    public void setDeg(String deg)
    {
        this.deg = deg;
    }

    public Department getDepartment()
    {
        return department;
    }

    public void setDepartment(Department department)
    {
        this.department = department;
    }
}
```

## Persistence.xml

persistence.xml文件，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persist
             http://java.sun.com/xml/ns/persistence/persistence_2_(
   <persistence-unit name="Eclipselink_JPA"
                          transaction-type="RESOURCE_LOCAL">
   <class>com.yiibai.eclipselink.entity.Employee</class>
   <class>com.yiibai.eclipselink.entity.Department</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                     value="jdbc:mysql://localhost:3306/jpadb"/>
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                            value="root"/>
           <property name="javax.persistence.jdbc.driver"
                     value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
           <property name="eclipselink.ddl-generation"
                     value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

## 服务类

在src下创建一个名为“com.yiibai.eclipselink.service'的包。在这个包下创建一个名为OneToOne.java的DAO类。在DAO类如下所示：

```
package com.yiibaieclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Department;
import com.yiibai.eclipselink.entity.Employee;

public class OneToOne
{
   public static void main(String[] args)
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );

       //Create Department Entity
       Department department = new Department();
       department.setName("Development");

       //Store Department
       entitymanager.persist(department);

       //Create Employee Entity
       Employee employee = new Employee();
       employee.setEname("Satish");
       employee.setSalary(45000.0);
       employee.setDeg("Technical Writer");
       employee.setDepartment(department);

       //Store Employee
       entitymanager.persist(employee);

       entitymanager.getTransaction().commit();
       entitymanager.close();
       emfactory.close();
   }
}
```

编译和执行上述程序后，在Eclipse IDE控制台面板的显示通知。对于输出，检查MySQL工作台如下。

在上面的例子中，将创建两个表。通过在MySQL的界面下面的查询，department表的结果显示如下：

```
Select * from department
```

| ID | Name |
| --- | --- |
| 301 | Development |

通过在MySQL界面下面查询，employee表的结果显示如下：

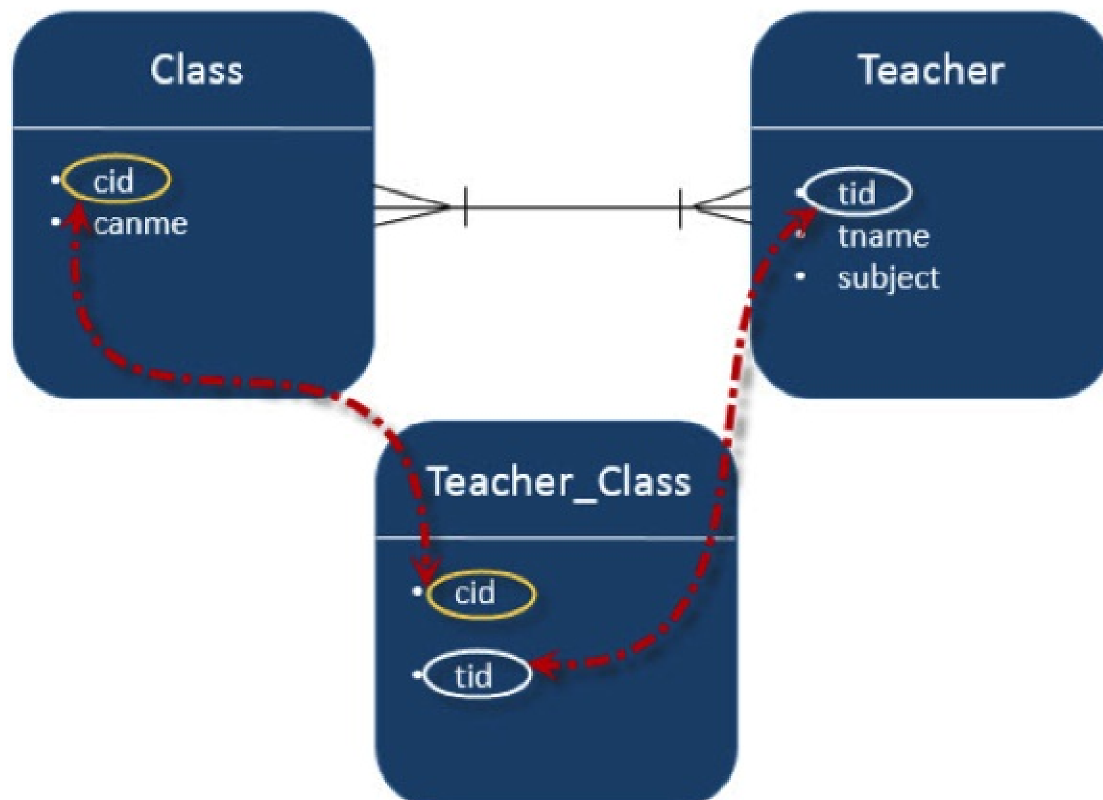
```
Select * from employee
```

| Eid | Deg | Ename | Salary | Department_id |
| --- | --- | --- | --- | --- |
| 302 | Technical Writer | Satish | 45000 | 301 |

## @ManyToMany 关系

多对多的关系，就是从一个实体中的一个或多个行与其他实体的多个行相关联。

让我们考虑两个实体之间的关系的一个例子：班级和教师。以双向的方式，既班级和教师有多对一的关系。这意味着类中的每个记录由教师组（教师ID），这应该是在教师表中的主键和存储在Teacher_Class表，反之亦然简称。在这里，Teachers_Class表包含外键字段。在Eclipse中创建一个IDE JPA项目命名JPA_Eclipselink_MTM。这个项目的所有模块下面讨论。



## 创建实体

按照上面的图中所示的方案中创建的实体。在src下创建一个名为“com.tutorialspoin.eclipselink.entity'的包。创建一个名为Class.java的类在这个包下。类系实体显示如下：

```
package com.yiibai.eclipselink.entity;

import java.util.Set;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

@Entity
public class Clas
{
   @Id
   @GeneratedValue( strategy = GenerationType.AUTO )
   private int cid;
   private String cname;

   @ManyToMany(targetEntity=Teacher.class)
   private Set teacherSet;

   public Clas()
   {
       super();
   }
   public Clas(int cid,
           String cname, Set teacherSet)
   {
       super();
       this.cid = cid;
       this.cname = cname;
       this.teacherSet = teacherSet;
   }
   public int getCid()
   {
       return cid;
   }
   public void setCid(int cid)
   {
       this.cid = cid;
   }
   public String getCname()
   {
       return cname;
   }
   public void setCname(String cname)
   {
```

```
        this.cname = cname;
    }
    public Set getTeacherSet()
    {
        return teacherSet;
    }
    public void setTeacherSet(Set teacherSet)
    {
        this.teacherSet = teacherSet;
    }
}
```

创建第二个实体这种关系的Employee实体类，命名为Teacher.java 在“com.yiibai.eclipselink.entity'包下。 Employee实体类如下所示：

```
package com.yiibai.eclipselink.entity;

import java.util.Set;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

@Entity
public class Teacher
{
   @Id
   @GeneratedValue( strategy = GenerationType.AUTO )
   private int tid;
   private String tname;
   private String subject;

   @ManyToMany(targetEntity=Clas.class)
   private Set clasSet;

   public Teacher()
   {
       super();
   }
   public Teacher(int tid, String tname, String subject,
           Set clasSet)
   {
       super();
       this.tid = tid;
       this.tname = tname;
       this.subject = subject;
       this.clasSet = clasSet;
   }
   public int getTid()
   {
```

```
        return tid;
    }
    public void setTid(int tid)
    {
        this.tid = tid;
    }
    public String getTname()
    {
        return tname;
    }
    public void setTname(String tname)
    {
        this.tname = tname;
    }
    public String getSubject()
    {
        return subject;
    }
    public void setSubject(String subject)
    {
        this.subject = subject;
    }
    public Set getClasSet()
    {
        return clasSet;
    }
    public void setClasSet(Set clasSet)
    {
        this.clasSet = clasSet;
    }
}
```

## Persistence.xml

Persistence.xml 文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persist
             http://java.sun.com/xml/ns/persistence/persistence_2_(
   <persistence-unit name="Eclipselink_JPA"
                          transaction-type="RESOURCE_LOCAL">
   <class>com.yiibai.eclipselink.entity.Employee</class>
   <class>com.yiibai.eclipselink.entity.Department</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                     value="jdbc:mysql://localhost:3306/jpadb"/>
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                            value="root"/>
           <property name="javax.persistence.jdbc.driver"
                     value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE",
           <property name="eclipselink.ddl-generation"
                     value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

## 服务类

在src下创建一个名为“com.yiibai.eclipselink.service'的包，在这个包下创建一个名为ManyToMany.java的DAO类。在DAO类如下所示：

```
package com.yiibai.eclipselink.service;

import java.util.HashSet;
import java.util.Set;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.yiibai.eclipselink.entity.Clas;
import com.yiibai.eclipselink.entity.Teacher;

public class ManyToMany
{
   public static void main(String[] args)
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
       EntityManager entitymanager = emfactory.
               createEntityManager( );
       entitymanager.getTransaction( ).begin( );
```

```
        //Create Clas Entity
        Clas clas1=new Clas(0,"1st",null);
        Clas clas2=new Clas(0,"2nd",null);
        Clas clas3=new Clas(0,"3rd",null);

        //Store Clas
        entitymanager.persist(clas1);
        entitymanager.persist(clas2);
        entitymanager.persist(clas3);

        //Create Clas Set1
        Set<Clas> classSet1 = new HashSet();
        classSet1.add(clas1);
        classSet1.add(clas2);
        classSet1.add(clas3);

        //Create Clas Set2
        Set<Clas> classSet2 = new HashSet();
        classSet2.add(clas3);
        classSet2.add(clas1);
        classSet2.add(clas2);

        //Create Clas Set3
        Set<Clas> classSet3 = new HashSet();
        classSet3.add(clas2);
        classSet3.add(clas3);
        classSet3.add(clas1);

        //Create Teacher Entity
        Teacher teacher1 = new Teacher(0,
                "Satish","Java",classSet1);
        Teacher teacher2 = new Teacher(0,
                "Krishna","Adv Java",classSet2);
        Teacher teacher3 = new Teacher(0,
                "Masthanvali","DB2",classSet3);

        //Store Teacher
        entitymanager.persist(teacher1);
        entitymanager.persist(teacher2);
        entitymanager.persist(teacher3);

        entitymanager.getTransaction( ).commit( );
        entitymanager.close( );
        emfactory.close( );
    }
}
```

在这个例子中工程，将创建三个表。通过在MySQL界面执行下面的查询，teacher_clas表将显示如下的结果：

```
Select * form teacher_clas
```

| Teacher_tid | Classet_cid |
|---|---|
| 354 | 351 |
| 355 | 351 |
| 356 | 351 |
| 354 | 352 |
| 355 | 352 |
| 356 | 352 |
| 354 | 353 |
| 355 | 353 |
| 356 | 353 |

在上表中teacher_tid是从eacher表的外键，classet_cid是class表的外键。因此不同的老师被分配到不同的班级。

通过在MySQL的界面下面的查询，teacher表的结果显示如下：

```
Select * from teacher
```

| Tid | Subject | Tname |
|---|---|---|
| 354 | Java | Satish |
| 355 | Adv Java | Krishna |
| 356 | DB2 | Masthanvali |

通过在MySQL的界面执行下面的查询, clas表将显示如下的结果：

```
Select * from clas
```

| Cid | Cname |
|---|---|
| 351 | 1st |
| 352 | 2nd |
| 353 | 3rd |

# JPA标准API - JPA教程

标准是用来定义查询实体的预定义API。它是定义JPQL查询的另一种方式。这些查询是类型安全的，可移植的，并且容易被改变的语法进行修改。类似于JPQL，它遵循的抽象模式(容易编辑模式)和嵌入的对象。元数据API是夹杂着标准的API模型持久性实体的标准查询。

标准的API的主要优点是，错误可以较早在编译时被检测到。基于字符串JPQL查询和基于查询JPA的范围是在性能和效率相同。

## 标准API历史

该标准被纳入因此标准的每一步中JPA的规范通知所有版本JPA。

- 在JPA2.0中，标准查询API，查询的标准化开发。
- 在JPA2.1，标准更新和删除（批量更新和删除）都包括在内。

## 标准查询结构

该标准与JPQL是密切相关的，并允许使用类似的操作符在他们的查询设计。它遵循javax.persistence.criteria包设计一个查询。查询结构指的语法条件查询。

下面简单的条件查询返回数据源中的实体类的所有实例。

```
EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Entity class> cq = cb.createQuery(Entity.class);
Root<Entity> from = cq.from(Entity.class);
cq.select(Entity);
TypedQuery<Entity> q = em.createQuery(cq);
List<Entity> allitems = q.getResultList();
```

查询演示了基本的步骤来创建一个标准。

- EntityManager实例被用来创建一个CriteriaBuilder对象。
- CriteriaQuery实例是用来创建一个查询对象。这个查询对象的属性将与该查询的细节进行修改。
- CriteriaQuery.form方法被调用来设置查询根。
- CriteriaQuery.select被调用来设置结果列表类型。
- TypedQuery<T>实例是用来准备一个查询执行和指定的查询结果的类型。

- 在TypedQuery<T>对象getResultList方法来执行查询。该查询返回实体的集合，结果存储在一个列表中。

# 标准API示例

让我们考虑 employee 数据库的例子。让我们假定该 jpadb.employee表包含以下记录：

```
Eid    Ename          Salary    Deg
401    Gopal            40000    Technical Manager
402    Manisha        40000    Proof reader
403    Masthanvali    35000    Technical Writer
404  Satish           30000    Technical writer
405    Krishna        30000    Technical Writer
406    Kiran            35000    Proof reader
```

在名为JPA_Eclipselink_Criteria Eclipse IDE中创建JPA项目。这个项目的所有模块下面讨论：

## 创建实体

在'src'中创建包名为com.yiibai.eclipselink.entity

创建一个名为Employee.java类。Employee实体如下图所示：

```
package com.yiibai.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Employee
{
   @Id
   @GeneratedValue(strategy= GenerationType.AUTO)
   private int eid;
   private String ename;
   private double salary;
   private String deg;
   public Employee(int eid, String ename, double salary, String deg
   {
       super( );
       this.eid = eid;
       this.ename = ename;
       this.salary = salary;
       this.deg = deg;
```

```
    }

    public Employee( )
    {
        super();
    }

    public int getEid( )
    {
        return eid;
    }
    public void setEid(int eid)
    {
        this.eid = eid;
    }

    public String getEname( )
    {
        return ename;
    }
    public void setEname(String ename)
    {
        this.ename = ename;
    }

    public double getSalary( )
    {
        return salary;
    }
    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    public String getDeg( )
    {
        return deg;
    }
    public void setDeg(String deg)
    {
        this.deg = deg;
    }
    @Override
    public String toString() {
        return "Employee [eid=" + eid + ", ename=" + ename + ", sala
                + salary + ", deg=" + deg + "]";
    }
}
```

## Persistence.xml

Persistence.xml 文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persis
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persist
             http://java.sun.com/xml/ns/persistence/persistence_2_0
   <persistence-unit name="Eclipselink_JPA"
                        transaction-type="RESOURCE_LOCAL">
   <class>com.yiibai.eclipselink.entity.Employee</class>
       <properties>
           <property name="javax.persistence.jdbc.url"
                     value="jdbc:mysql://localhost:3306/jpadb"/>
           <property name="javax.persistence.jdbc.user" value="root
           <property name="javax.persistence.jdbc.password"
                     value="root"/>
           <property name="javax.persistence.jdbc.driver"
                     value="com.mysql.jdbc.Driver"/>
           <property name="eclipselink.logging.level" value="FINE"/
           <property name="eclipselink.ddl-generation"
                     value="create-tables"/>
       </properties>
   </persistence-unit>
</persistence>
```

## 服务类

该模块包含服务类，它实现了使用元数据API的初始化条件查询的一部分。创建一个名为“com.yiibai.eclipselink.service'包。在这个包下创建一个类CriteriaAPI.java。在DAO类如下所示：

```
package com.yiibai.eclipselink.service;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import com.yiibai.eclipselink.entity.Employee;

public class CriteriaApi
{
   public static void main(String[] args)
   {
       EntityManagerFactory emfactory = Persistence.
               createEntityManagerFactory( "Eclipselink_JPA" );
```

```
        EntityManager entitymanager = emfactory.
                createEntityManager( );
        CriteriaBuilder criteriaBuilder = entitymanager
                .getCriteriaBuilder();
        CriteriaQuery<Object> criteriaQuery = criteriaBuilder
                .createQuery();
        Root<Employee> from = criteriaQuery.from(Employee.class);

        //select all records
         System.out.println(“Select all records”);
        CriteriaQuery<Object> select =criteriaQuery.select(from);
        TypedQuery<Object> typedQuery = entitymanager
                .createQuery(select);
        List<Object> resultlist= typedQuery.getResultList();

        for(Object o:resultlist)
        {
            Employee e=(Employee)o;
            System.out.println("EID : "+e.getEid()
                    +" Ename : "+e.getEname());
        }

        //Ordering the records
         System.out.println(“Select all records by follow ordering”)
        CriteriaQuery<Object> select1 = criteriaQuery.select(from);
         select1.orderBy(criteriaBuilder.asc(from.get("ename")));
         TypedQuery<Object> typedQuery1 = entitymanager
                .createQuery(select);
         List<Object> resultlist1= typedQuery1.getResultList();

        for(Object o:resultlist1)
        {
            Employee e=(Employee)o;
            System.out.println("EID : "+e.getEid()
                    +" Ename : "+e.getEname());
        }

        entitymanager.close( );
        emfactory.close( );
    }
}
```

编译和执行上述程序后，将在Eclipse IDE的控制台面板输出下面的内容。

```
Select All records
EID : 401 Ename : Gopal
EID : 402 Ename : Manisha
EID : 403 Ename : Masthanvali
EID : 404 Ename : Satish
EID : 405 Ename : Krishna
EID : 406 Ename : Kiran
Select All records by follow Ordering
EID : 401 Ename : Gopal
EID : 406 Ename : Kiran
EID : 405 Ename : Krishna
EID : 402 Ename : Manisha
EID : 403 Ename : Masthanvali
EID : 404 Ename : Satish
```

# JSP 教程

# JSP 基础

# JSP 简介

## 什么是Java Server Pages?

JSP全称Java Server Pages，是一种动态网页开发技术。它使用JSP标签在HTML网页中插入Java代码。标签通常以<%开头以%>结束。

JSP是一种Java servlet，主要用于实现Java web应用程序的用户界面部分。网页开发者们通过结合HTML代码、XHTML代码、XML元素以及嵌入JSP操作和命令来编写JSP。

JSP通过网页表单获取用户输入数据、访问数据库及其他数据源，然后动态地创建网页。

JSP标签有多种功能，比如访问数据库、记录用户选择信息、访问JavaBeans组件等，还可以在不同的网页中传递控制信息和共享信息。

## 为什么使用JSP？

JSP程序与CGI程序有着相似的功能，但和CGI程序相比，JSP程序有如下优势：

- 性能更加优越，因为JSP可以直接在HTML网页中动态嵌入元素而不需要单独引用CGI文件。
- 服务器调用的是已经编译好的JSP文件，而不像CGI/Perl那样必须先载入解释器和目标脚本。
- JSP基于Java Servlets API，因此，JSP拥有各种强大的企业级Java API，包括JDBC，JNDI，EJB，JAXP等等。
- JSP页面可以与处理业务逻辑的servlets一起使用，这种模式被Java servlet 模板引擎所支持。

最后，JSP是Java EE不可或缺的一部分，是一个完整的企业级应用平台。这意味着JSP可以用最简单的方式来实现最复杂的应用。

## JSP的优势

以下列出了使用JSP带来的其他好处：

- 与ASP相比：JSP有两大优势。首先，动态部分用Java编写，而不是VB或其他MS专用语言，所以更加强大与易用。第二点就是JSP易于移植到非MS平台上。
- 与纯 Servlets相比：JSP可以很方便的编写或者修改HTML网页而不用去面对大量的println语句。
- 与SSI相比：SSI无法使用表单数据、无法进行数据库链接。
- 与JavaScript相比：虽然JavaScript可以在客户端动态生成HTML，但是很难与

服务器交互，因此不能提供复杂的服务，比如访问数据库和图像处理等等。
- 与静态HTML相比：静态HTML不包含动态信息。

## 接下来呢？

我们将会带您一步一步地来搭建JSP运行环境，这需要有一定的Java基础。

如果您还未学过Java，可以先学习我们为您提供的Java教程。

# JSP 开发环境搭建

JSP开发环境是您用来开发、测试和运行JSP程序的地方。

本节将会带您搭建JSP开发环境，具体包括以下几个步骤。

## 配置Java开发工具（JDK）

这一步涉及Java SDK的下载和PATH环境变量的配置。

您可以从Oracle公司的Java页面中下载SDK：Java SE Downloads

Java SDK下载完后，请按照给定的指示来安装和配置SDK。最后，通过设置PATH和JAVA_HOME环境变量来指明包括java和javac的文件夹路径，通常是java_install_dir/bin和java_install_dir。

假如您用的是Windows系统并且SDK的安装目录为C::\jdk1.5.0_20，那么您就需要在 C:\autoexec.bat 文件中添加以下两行：

```
set PATH=C:\jdk1.5.0_20\bin;%PATH%
set JAVA_HOME=C:\jdk1.5.0_20
```

或者，在Windows NT/2000/XP下，您可以直接右击我的电脑图标，选择属性，然后高级，然后环境变量，接下来您就可以很方便地设置PATH变量并且确定退出就行了。

在Linux/Unix系统下，如果SDK的安装目录为/usr/local/jdk1.5.0_20并且使用的是C shell，那么您就需要在.cshrc文件中添加以下两行：

```
setenv PATH /usr/local/jdk1.5.0_20/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.5.0_20
```

或者，假如您正在使用类似于Borland JBuilder、Eclipse、IntelliJ IDEA和Sun ONE Studio这样的集成开发环境，可以试着编译并运行一个简单的程序来确定IDE（集成开发环境）是否已经知道 SDK的安装目录。

本步骤你也可以参考本站Java开发环境配置章节的教程。

## 设置Web服务器：Tomcat

目前，市场上有很多支持JSP和Servlets开发的Web服务器。他们中的一些可以免费下载和使用，Tomcat就是其中之一。

Apache Tomcat是一个开源软件，可作为独立的服务器来运行JSP和Servlets，也可以集成在 Apache Web Server中。以下是Tomcat的配置方法：

- 下载最新版本的Tomcat：http://tomcat.apache.org/。
- 下载完安装文件后，将压缩文件解压到一个方便的地方，比如Windows下的C:\apache-tomcat-5.5.29目录或者Linux/Unix下的/usr/local/apache-tomcat-5.5.29目录，然后创建CATALINA_HOME环境变量指向这些目录。

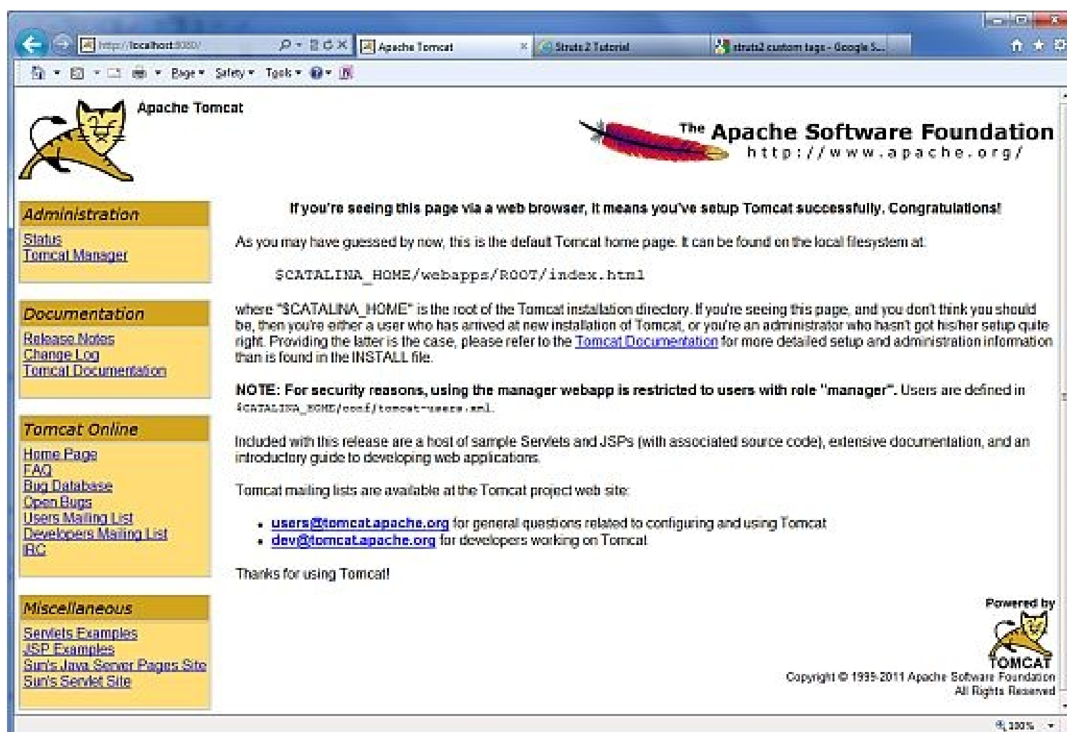
在Windows机器下，Tomcat可以通过执行以下命令来启动：

```
%CATALINA_HOME%\bin\startup.bat
或者
C:\apache-tomcat-5.5.29\bin\startup.bat
```

在Linux/Unix机器下，Tomcat可以通过执行以下命令来启动：

```
$CATALINA_HOME/bin/startup.sh
或者
/usr/local/apache-tomcat-5.5.29/bin/startup.sh
```

成功启动Tomcat后，通过访问http://localhost:8080/便可以使用Tomcat自带的一些web应用了。假如一切顺利的话，您应该能够看到以下的页面：




更多关于配置和运行Tomcat的信息可以在Tomcat提供的文档中找到，或者去Tomcat官网查阅：http://tomcat.apache.org。

在Windows机器下，Tomcat可以通过执行以下命令来停止：

```
%CATALINA_HOME%\bin\shutdown
或者
C:\apache-tomcat-5.5.29\bin\shutdown
```

在Linux/Unix机器下，Tomcat可以通过执行以下命令来停止：

```
$CATALINA_HOME/bin/shutdown.sh
或者
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

# 设置CLASSPATH环境变量

由于servlets不是Java SE的一部分，所以您必须标示出servlet类的编译器。

假如您用的是Windows机器，您需要在C:\autoexec.bat文件中添加以下两行：

```
set CATALINA=C:\apache-tomcat-5.5.29
set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%
```

或者，在Windows NT/2000/XP下，您只要右击我的电脑，选择属性，然后点击高级，然后点击环境变量，接下来便可以设置CLASSPATH变量并且确定退出即可。

在Linux/Unix机器下，假如您使用的是C shell，那么您就需要在.cshrc文件中添加以下两行：

```
setenv CATALINA=/usr/local/apache-tomcat-5.5.29
setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH
```

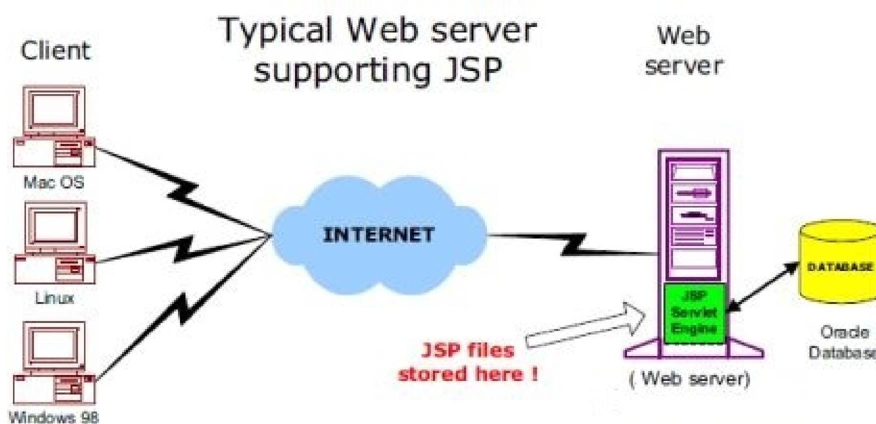
注意：如果您的开发路径是C:\JSPDev (Windows)或者 /usr/JSPDev (Linux/Unix)，那么您就需要将这些路径添加进CLASSPATH变量中。

# JSP 结构

网络服务器需要一个JSP引擎，也就是一个容器来处理JSP页面。容器负责截获对JSP页面的请求。本教程使用内嵌JSP容器的Apache来支持JSP开发。

JSP容器与Web服务器协同合作，为JSP的正常运行提供必要的运行环境和其他服务，并且能够正确识别专属于JSP网页的特殊元素。

下图显示了JSP容器和JSP文件在Web应用中所处的位置。

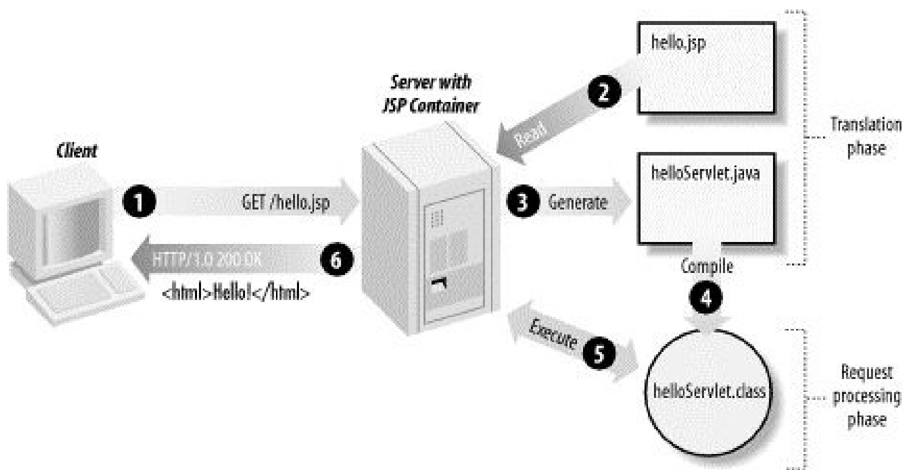



## JSP处理

以下步骤表明了Web服务器是如何使用JSP来创建网页的：

- 就像其他普通的网页一样，您的浏览器发送一个HTTP请求给服务器。
- Web服务器识别出这是一个对JSP网页的请求，并且将该请求传递给JSP引擎。通过使用URL或者.jsp文件来完成。
- JSP引擎从磁盘中载入JSP文件，然后将它们转化为servlet。这种转化只是简单地将所有模板文本改用println()语句，并且将所有的JSP元素转化成Java代码。
- JSP引擎将servlet编译成可执行类，并且将原始请求传递给servlet引擎。
- Web服务器的某组件将会调用servlet引擎，然后载入并执行servlet类。在执行过程中，servlet产生HTML格式的输出并将其内嵌于HTTP response中上交给Web服务器。
- Web服务器以静态HTML网页的形式将HTTP response返回到您的浏览器中。
- 最终，Web浏览器处理HTTP response中动态产生的HTML网页，就好像在处理静态网页一样。

以上提及到的步骤可以用下图来表示：




一般情况下，JSP引擎会检查JSP文件对应的servlet是否已经存在，并且检查JSP文件的修改日期是否早于servlet。如果JSP文件的修改日期早于对应的servlet，那么容器就可以确定JSP文件没有被修改过并且servlet有效。这使得整个流程与其他脚本语言（比如PHP）相比要高效快捷一些。

总的来说，JSP网页就是用另一种方式来编写servlet而不用成为Java编程高手。除了解释阶段外，JSP网页几乎可以被当成一个普通的servlet来对待。

# JSP 生命周期

理解JSP底层功能的关键就是去理解它们所遵守的生命周期。

JSP生命周期就是从创建到销毁的整个过程，类似于servlet生命周期，区别在于JSP生命周期还包括将JSP文件编译成servlet。

以下是JSP生命周期中所走过的几个阶段：

- 编译阶段：

  servlet容器编译servlet源文件，生成servlet类

- 初始化阶段：

  加载与JSP对应的servlet类，创建其实例，并调用它的初始化方法

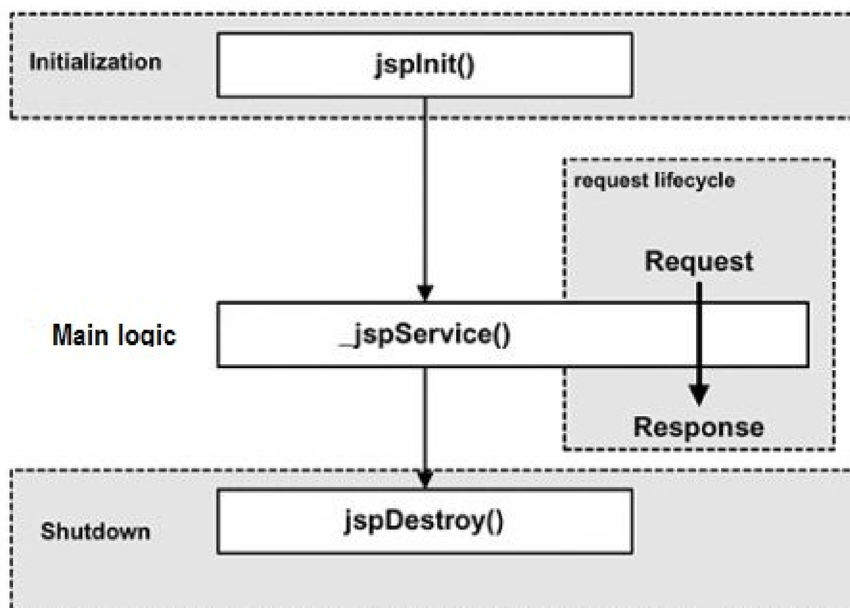
- 执行阶段：

  调用与JSP对应的servlet实例的服务方法

- 销毁阶段：

  调用与JSP对应的servlet实例的销毁方法，然后销毁servlet实例

很明显，JSP生命周期的四个主要阶段和servlet生命周期非常相似，下面给出图示：




# JSP编译

当浏览器请求JSP页面时，JSP引擎会首先去检查是否需要编译这个文件。如果这个文件没有被编译过，或者在上次编译后被更改过，则编译这个JSP文件。

编译的过程包括三个步骤：

- 解析JSP文件。
- 将JSP文件转为servlet。
- 编译servlet。

# JSP初始化

容器载入JSP文件后，它会在为请求提供任何服务前调用jspInit()方法。如果您需要执行自定义的JSP初始化任务，复写jspInit()方法就行了，就像下面这样：

```
public void jspInit(){
  // 初始化代码
}
```

一般来讲程序只初始化一次，servlet也是如此。通常情况下您可以在jspInit()方法中初始化数据库连接、打开文件和创建查询表。

# JSP执行

这一阶段描述了JSP生命周期中一切与请求相关的交互行为，直到被销毁。

当JSP网页完成初始化后，JSP引擎将会调用_jspService()方法。

_jspService()方法需要一个HttpServletRequest对象和一个HttpServletResponse对象作为它的参数，就像下面这样：

```
void _jspService(HttpServletRequest request,
                 HttpServletResponse response)
{
   // 服务端处理代码
}
```

_jspService()方法在每个request中被调用一次并且负责产生与之相对应的response，并且它还负责产生所有7个HTTP方法的回应，比如GET、POST、DELETE等等。

# JSP清理

JSP生命周期的销毁阶段描述了当一个JSP网页从容器中被移除时所发生的一切。

jspDestroy()方法在JSP中等价于servlet中的销毁方法。当您需要执行任何清理工作时复写jspDestroy()方法，比如释放数据库连接或者关闭文件夹等等。

jspDestroy()方法的格式如下：

```
public void jspDestroy()
{
   // 清理代码
}
```

## 实例

JSP生命周期代码实例如下所示：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html><head><title>life.jsp</title></head><body>

<%!
  private int initVar=0;
  private int serviceVar=0;
  private int destroyVar=0;
%>

<%!
  public void jspInit(){
    initVar++;
    System.out.println("jspInit(): JSP被初始化了"+initVar+"次");
  }
  public void jspDestroy(){
    destroyVar++;
    System.out.println("jspDestroy(): JSP被销毁了"+destroyVar+"次");
  }
%>

<%
  serviceVar++;
  System.out.println("_jspService(): JSP共响应了"+serviceVar+"次请求"

  String content1="初始化次数 : "+initVar;
  String content2="响应客户请求次数 : "+serviceVar;
  String content3="销毁次数 : "+destroyVar;
%>

<h1><%=content1 %></h1>
<h1><%=content2 %></h1>
<h1><%=content3 %></h1>

</body></html>
```

# JSP 语法

本小节将会简单地介绍一下JSP开发中的基础语法。

## 脚本程序

脚本程序可以包含任意量的Java语句、变量、方法或表达式，只要它们在脚本语言中是有效的。

脚本程序的语法格式：

```
<% 代码片段 %>
```

或者，您也可以编写与其等价的XML语句，就像下面这样：

```
<jsp:scriptlet>
   代码片段
</jsp:scriptlet>
```

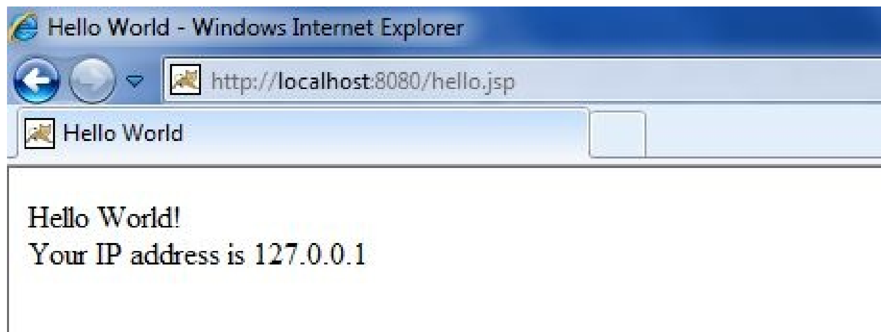
任何文本、HTML标签、JSP元素必须写在脚本程序的外面。

下面给出一个示例，同时也是本教程的第一个JSP示例：

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>
</body>
</html>
```

注意：请确保Apache Tomcat已经安装在C:\apache-tomcat-7.0.2目录下并且运行环境已经正确设置。

将以上代码保存在hello.jsp中，然后将它放置在 C:\apache-tomcat-7.0.2\webapps\ROOT目录下，打开浏览器并在地址栏中输入 http://localhost:8080/hello.jsp。运行后得到以下结果：




# JSP声明

一个声明语句可以声明一个或多个变量、方法，供后面的Java代码使用。在JSP文件中，您必须先声明这些变量和方法然后才能使用它们。

JSP声明的语法格式：

```
<%! declaration; [ declaration; ]+ ... %>
```

或者，您也可以编写与其等价的XML语句，就像下面这样：

```
<jsp:declaration>
   代码片段
</jsp:declaration>
```

程序示例：

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

# JSP表达式

一个JSP表达式中包含的脚本语言表达式，先被转化成String，然后插入到表达式出现的地方。

由于表达式的值会被转化成String，所以您可以在一个文本行中使用表达式而不用去管它是否是HTML标签。

表达式元素中可以包含任何符合Java语言规范的表达式，但是不能使用分号来结束表达式。

JSP表达式的语法格式：

```
<%= 表达式 %>
```

同样，您也可以编写与之等价的XML语句：

```
<jsp:expression>
   表达式
</jsp:expression>
```

程序示例：

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

运行后得到以下结果：

```
Today's date: 11-Sep-2013 21:24:25
```

# JSP注释

JSP注释主要有两个作用：为代码作注释以及将某段代码注释掉。

JSP注释的语法格式：

```
<%-- 这里可以填写 JSP 注释 --%>
```

程序示例：

```
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- 该部分注释在网页中不会被显示--%>
</body>
</html>
```

运行后得到以下结果：

```
A Test of Comments
```

不同情况下使用注释的语法规则：

| 语法 | 描述 |
|---|---|
| <%-- 注释 --%> | JSP注释，注释内容不会被发送至浏览器甚至不会被编译 |
| <!-- 注释 --> | HTML注释，通过浏览器查看网页源代码时可以看见注释内容 |
| <\% | 代表静态 <%常量 |
| %\> | 代表静态 %> 常量 |
| \' | 在属性中使用的单引号 |
| \" | 在属性中使用的双引号 |

# JSP指令

JSP指令用来设置与整个JSP页面相关的属性。

JSP指令语法格式：

```
<%@ directive attribute="value" %>
```

这里有三种指令标签：

| 指令 | 描述 |
|---|---|
| <%@ page ... %> | 定义页面的依赖属性，比如脚本语言、error页面、缓存需求等等 |
| <%@ include ... %> | 包含其他文件 |
| <%@ taglib ... %> | 引入标签库的定义，可以是自定义标签 |

# JSP行为

JSP行为标签使用XML语法结构来控制servlet引擎。它能够动态插入一个文件，重用JavaBean组件，引导用户去另一个页面，为Java插件产生相关的HTML等等。

行为标签只有一种语法格式，它严格遵守XML标准：

```
<jsp:action_name attribute="value" />
```

行为标签基本上是一些预先就定义好的函数，下表罗列出了一些可用的JSP行为标签：：

| 语法 | 描述 |
|---|---|
| jsp:include | 用于在当前页面中包含静态或动态资源 |
| jsp:useBean | 寻找和初始化一个JavaBean组件 |
| jsp:setProperty | 设置 JavaBean组件的值 |
| jsp:getProperty | 将 JavaBean组件的值插入到 output中 |
| jsp:forward | 从一个JSP文件向另一个文件传递一个包含用户请求的request对象 |
| jsp:plugin | 用于在生成的HTML页面中包含Applet和JavaBean对象 |
| jsp:element | 动态创建一个XML元素 |
| jsp:attribute | 定义动态创建的XML元素的属性 |
| jsp:body | 定义动态创建的XML元素的主体 |
| jsp:text | 用于封装模板数据 |

# JSP隐含对象

JSP支持九个自动定义的变量，江湖人称隐含对象。这九个隐含对象的简介见下表：

| 对象 | 描述 |
| --- | --- |
| request | **HttpServletRequest**类的实例 |
| response | **HttpServletResponse**类的实例 |
| out | **PrintWriter**类的实例，用于把结果输出至网页上 |
| session | **HttpSession**类的实例 |
| application | **ServletContext**类的实例，与应用上下文有关 |
| config | **ServletConfig**类的实例 |
| pageContext | **PageContext**类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | **Exception**类的对象，代表发生错误的JSP页面中对应的异常对象 |

# 控制流语句

JSP提供对Java语言的全面支持。您可以在JSP程序中使用Java API甚至建立Java代码块，包括判断语句和循环语句等等。

# 判断语句

If…else块，请看下面这个例子：

```
<%! int day = 3; %>
<html>
<head><title>IF...ELSE Example</title></head>
<body>
<% if (day == 1 | day == 7) { %>
      <p> Today is weekend</p>
<% } else { %>
      <p> Today is not weekend</p>
<% } %>
</body>
</html>
```

运行后得到以下结果：

```
Today is not weekend
```

现在来看看switch…case块，与if…else块有很大的不同，它使用out.println()，并且整个都装在脚本程序的标签中，就像下面这样：

```
<%! int day = 3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
<%
switch(day) {
case 0:
   out.println("It\'s Sunday.");
   break;
case 1:
   out.println("It\'s Monday.");
   break;
case 2:
   out.println("It\'s Tuesday.");
   break;
case 3:
   out.println("It\'s Wednesday.");
   break;
case 4:
   out.println("It\'s Thursday.");
   break;
case 5:
   out.println("It\'s Friday.");
   break;
default:
   out.println("It's Saturday.");
}
%>
</body>
</html>
```

运行后得出以下结果：

```
It's Wednesday.
```

循环语句

在JSP程序中可以使用Java的三个基本循环类型：for，while，和 do…while。

让我们来看看for循环的例子：

```
<%! int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body>
<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
   <font color="green" size="<%= fontSize %>">
    JSP Tutorial
   </font><br />
<%}%>
</body>
</html>
```

运行后得到以下结果：

JSP Tutorial
JSP Tutorial
JSP Tutorial

将上例改用while循环来写：

```
<%! int fontSize; %>
<html>
<head><title>WHILE LOOP Example</title></head>
<body>
<%while ( fontSize <= 3){ %>
   <font color="green" size="<%= fontSize %>">
    JSP Tutorial
   </font><br />
<%fontSize++;%>
<%}%>
</body>
</html>
```

运行后得到同样的结果：
JSP Tutorial
JSP Tutorial
JSP Tutorial

# JSP运算符

JSP支持所有Java逻辑和算术运算符。

下表罗列出了JSP常见运算符，优先级从高到底：

| 类别 | 操作符 | 结合性 |
|---|---|---|
| 后缀 | () [] . (点运算符) | 左到右 |
| 一元 | ++ - - ! ~ | 右到左 |
| 可乘性 | * / % | 左到右 |
| 可加性 | + - | 左到右 |
| 移位 | >> >>> << | 左到右 |
| 关系 | > >= < <= | 左到右 |
| 相等/不等 | == != | 左到右 |
| 位与 | & | 左到右 |
| 位异或 | ^ | 左到右 |
| 位或 | \| | 左到右 |
| 逻辑与 | && | 左到右 |
| 逻辑或 | \|\| | 左到右 |
| 条件判断 | ?: | 右到左 |
| 赋值 | = += -= *= /= %= >>= <<= &= ^= \|= | 右到左 |
| 逗号 | , | 左到右 |

# JSP常量

JSP语言定义了以下几个常量：

- Boolean：true and false
- Integer：与Java中的一样
- Floating point：与Java中的一样
- String：以单引号或双引号开始和结束。 " 被转义成 \"，'被转义成 \'， \ 被转义成\
- Null：null

# JSP 指令

JSP指令用来设置整个JSP页面相关的属性，如网页的编码方式和脚本语言。

语法格式如下：

```
<%@ directive attribute="value" %>
```

指令可以有很多个属性，它们以键值对的形式存在，并用逗号隔开。

JSP中的三种指令标签：

| 指令 | 描述 |
| --- | --- |
| <%@ page ... %> | 定义网页依赖属性，比如脚本语言、error页面、缓存需求等等 |
| <%@ include ... %> | 包含其他文件 |
| <%@ taglib ... %> | 引入标签库的定义 |

## Page指令

Page指令为容器提供当前页面的使用说明。一个JSP页面可以包含多个page指令。

Page指令的语法格式：

```
<%@ page attribute="value" %>
```

等价的XML格式：

```
<jsp:directive.page attribute="value" />
```

## 属性

下表列出与Page指令相关的属性：

| 属性 | 描述 |
| --- | --- |
| buffer | 指定out对象使用缓冲区的大小 |
| autoFlush | 控制out对象的 缓存区 |
| contentType | 指定当前JSP页面的MIME类型和字符编码 |
| errorPage | 指定当JSP页面发生异常时需要转向的错误处理页面 |
| isErrorPage | 指定当前页面是否可以作为另一个JSP页面的错误处理页面 |
| extends | 指定servlet从哪一个类继承 |
| import | 导入要使用的Java类 |
| info | 定义JSP页面的描述信息 |
| isThreadSafe | 指定对JSP页面的访问是否为线程安全 |
| language | 定义JSP页面所用的脚本语言，默认是Java |
| session | 指定JSP页面是否使用session |
| isELIgnored | 指定是否执行EL表达式 |
| isScriptingEnabled | 确定脚本元素能否被使用 |

# Include指令

JSP可以通过include指令来包含其他文件。被包含的文件可以是JSP文件、HTML文件或文本文件。包含的文件就好像是该JSP文件的一部分，会被同时编译执行。

Include指令的语法格式如下：

```
<%@ include file="relative url" %>
```

Include指令中的文件名实际上是一个相对的URL。如果您没有给文件关联一个路径，JSP编译器默认在当前路径下寻找。

等价的XML语法：

```
<jsp:directive.include file="relative url" />
```

# Taglib指令

JSP API允许用户自定义标签，一个自定义标签库就是自定义标签的集合。

Taglib指令引入一个自定义标签集合的定义，包括库路径、自定义标签。

Taglib指令的语法：

```
<%@ taglib uri="uri" prefix="prefixOfTag" %>
```

uri属性确定标签库的位置，prefix属性指定标签库的前缀。

等价的XML语法：

```
<jsp:directive.taglib uri="uri" prefix="prefixOfTag" />
```

# JSP 动作元素

与JSP指令元素不同的是，JSP动作元素在请求处理阶段起作用。JSP动作元素是用XML语法写成的。

利用JSP动作可以动态地插入文件、重用JavaBean组件、把用户重定向到另外的页面、为Java插件生成HTML代码。

动作元素只有一种语法，它符合XML标准：

```
<jsp:action_name attribute="value" />
```

动作元素基本上都是预定义的函数，JSP规范定义了一系列的标准动作，它用JSP作为前缀，可用的标准动作元素如下：

| 语法 | 描述 |
| --- | --- |
| jsp:include | 在页面被请求的时候引入一个文件。 |
| jsp:useBean | 寻找或者实例化一个JavaBean。 |
| jsp:setProperty | 设置JavaBean的属性。 |
| jsp:getProperty | 输出某个JavaBean的属性。 |
| jsp:forward | 把请求转到一个新的页面。 |
| jsp:plugin | 根据浏览器类型为Java插件生成OBJECT或EMBED标记。 |
| jsp:element | 定义动态XML元素 |
| jsp:attribute | 设置动态定义的XML元素属性。 |
| jsp:body | 设置动态定义的XML元素内容。 |
| jsp:text | 在JSP页面和文档中使用写入文本的模板 |

## 常见的属性

所有的动作要素都有两个属性：id属性和scope属性。

- **id**属性：

  id属性是动作元素的唯一标识，可以在JSP页面中引用。动作元素创建的id值可以通过PageContext来调用。

- **scope**属性：

该属性用于识别动作元素的生命周期。 id属性和scope属性有直接关系，scope属性定义了相关联id对象的寿命。 scope属性有四个可能的值： (a) page, (b)request, (c)session, 和 (d) application。

# <jsp:include>动作元素

<jsp:include>动作元素用来包含静态和动态的文件。该动作把指定文件插入正在生成的页面。语法格式如下：

```
<jsp:include page="relative URL" flush="true" />
```

前面已经介绍过include指令，它是在JSP文件被转换成Servlet的时候引入文件，而这里的jsp:include动作不同，插入文件的时间是在页面被请求的时候。

以下是include动作相关的属性列表。

| 属性 | 描述 |
|---|---|
| page | 包含在页面中的相对URL地址。 |
| flush | 布尔属性，定义在包含资源前是否刷新缓存区。 |

## 实例

以下我们定义了两个文件date.jsp和main.jsp，代码如下所示：

date.jsp文件代码：

```
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

main.jsp文件代码：

```
<html>
<head>
<title>The include Action Example</title>
</head>
<body>
<center>
<h2>The include action Example</h2>
<jsp:include page="date.jsp" flush="true" />
</center>
</body>
</html>
```

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

```
The include action Example
Today's date: 12-Sep-2013 14:54:22
```

# <jsp:useBean>动作元素

jsp:useBean动作用来装载一个将在JSP页面中使用的JavaBean。

这个功能非常有用，因为它使得我们既可以发挥Java组件重用的优势，同时也避免了损失JSP区别于Servlet的方便性。

jsp:useBean动作最简单的语法为：

```
<jsp:useBean id="name" class="package.class" />
```

在类载入后，我们既可以通过 jsp:setProperty 和 jsp:getProperty 动作来修改和检索bean的属性。

以下是useBean动作相关的属性列表。

| 属性 | 描述 |
|---|---|
| class | 指定Bean的完整包名。 |
| type | 指定将引用该对象变量的类型。 |
| beanName | 通过 java.beans.Beans 的 instantiate() 方法指定Bean的名字。 |

在给出具体实例前，让我们先来看下 jsp:setProperty 和 jsp:getProperty 动作元素：

# <jsp:setProperty>动作元素

jsp:setProperty用来设置已经实例化的Bean对象的属性，有两种用法。首先，你可以在jsp:useBean元素的外面（后面）使用jsp:setProperty，如下所示：

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName" property="someProperty" .../>
```

此时，不管jsp:useBean是找到了一个现有的Bean，还是新创建了一个Bean实例，jsp:setProperty都会执行。第二种用法是把jsp:setProperty放入jsp:useBean元素的内部，如下所示：

```
<jsp:useBean id="myName" ... >
...
   <jsp:setProperty name="myName" property="someProperty" .../>
</jsp:useBean>
```

此时，jsp:setProperty只有在新建Bean实例时才会执行，如果是使用现有实例则不执行jsp:setProperty。

jsp:setproperty动作有下面四个属性,如下表：

| 属性 | 描述 |
|---|---|
| name | name属性是必需的。它表示要设置属性的是哪个Bean。 |
| property | property属性是必需的。它表示要设置哪个属性。有一个特殊用法：如果property的值是"*"，表示所有名字和Bean属性名字匹配的请求参数都将被传递给相应的属性set方法。 |
| value | value 属性是可选的。该属性用来指定Bean属性的值。字符串数据会在目标类中通过标准的valueOf方法自动转换成数字、boolean、Boolean、 byte、Byte、char、Character。例如，boolean和Boolean类型的属性值（比如"true"）通过 Boolean.valueOf转换，int和Integer类型的属性值（比如"42"）通过Integer.valueOf转换。 value和param不能同时使用，但可以使用其中任意一个。 |
| param | param 是可选的。它指定用哪个请求参数作为Bean属性的值。如果当前请求没有参数，则什么事情也不做，系统不会把null传递给Bean属性的set方法。因此，你可以让Bean自己提供默认属性值，只有当请求参数明确指定了新值时才修改默认属性值。 |

# <jsp:getProperty>动作元素

jsp:getProperty动作提取指定Bean属性的值，转换成字符串，然后输出。语法格式如下：

```
<jsp:useBean id="myName" ... />
...
<jsp:getProperty name="myName" property="someProperty" .../>
```

下表是与getProperty相关联的属性：

| 属性 | 描述 |
|---|---|
| name | 要检索的Bean属性名称。Bean必须已定义。 |
| property | 表示要提取Bean属性的值 |

## 实例

以下实例我们使用了Bean:

```
/* 文件: TestBean.java */
package action;

public class TestBean {
   private String message = "No message specified";

   public String getMessage() {
      return(message);
   }
   public void setMessage(String message) {
      this.message = message;
   }
}
```

编译以上实例并生成 TestBean.class 文件，将该文件拷贝至服务器正式存放Java类的目录下，而不是保留给修改后能够自动装载的类的目录( 如：C:\apache-tomcat-7.0.2\webapps\WEB-INF\classes\action目录中，CLASSPATH 变量必须包含该路径。 )。例如，对于Java Web Server来说，Bean和所有Bean用到的类都应该放入classes目录，或者封装进jar文件后放入lib目录，但不应该放到servlets 下。
下面是一个很简单的例子，它的功能是装载一个Bean，然后设置/读取它的message属性。

现在让我们在main.jsp文件中调用该Bean:

```
 <html>
<head>
<title>Using JavaBeans in JSP</title>
</head>
<body>
<center>
<h2>Using JavaBeans in JSP</h2>

<jsp:useBean id="test" class="action.TestBean" />

<jsp:setProperty name="test"
                    property="message"
                    value="Hello JSP..." />

<p>Got message....</p>

<jsp:getProperty name="test" property="message" />

</center>
</body>
</html>
```

执行以上文件，输出如下所示：

```
Using JavaBeans in JSP
Got message....
Hello JSP...
```

# <jsp:forward> 动作元素

jsp:forward动作把请求转到另外的页面。jsp:forward标记只有一个属性page。语法格式如下所示：

```
<jsp:forward page="Relative URL" />
```

以下是forward相关联的属性：

| 属性 | 描述 |
| --- | --- |
| page | page属性包含的是一个相对URL。page的值既可以直接给出，也可以在请求的时候动态计算，可以是一个JSP页面或者一个 Java Servlet. |

## 实例

以下实例我们使用了两个文件，分别是：date.jps 和 main.jsp。

date.js文件代码如下：

```
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

main.jsp文件代码：

```
<html>
<head>
<title>The forward Action Example</title>
</head>
<body>
<center>
<h2>The forward action Example</h2>
<jsp:forward page="date.jsp" />
</center>
</body>
```

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

```
Today's date: 12-Sep-2010 14:54:22
```

## <jsp:plugin>动作元素

jsp:plugin动作用来根据浏览器的类型，插入通过Java插件 运行Java Applet所必需的OBJECT或EMBED元素。

如果需要的插件不存在，它会下载插件，然后执行Java组件。 Java组件可以是一个applet或一个JavaBean。

plugin动作有多个对应HTML元素的属性用于格式化Java 组件。param元素可用于向Applet 或 Bean 传递参数。

以下是使用plugin 动作元素的典型实例:

```
<jsp:plugin type="applet" codebase="dirname" code="MyApplet.class"
                           width="60" height="80">
   <jsp:param name="fontcolor" value="red" />
   <jsp:param name="background" value="black" />

   <jsp:fallback>
      Unable to initialize Java Plugin
   </jsp:fallback>

</jsp:plugin>
```

如果你有兴趣可以尝试使用applet来测试jsp:plugin动作元素，<fallback>元素是一个新元素，在组件出现故障的错误是发送给用户错误信息。

## <jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素

<jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素动态定义XML元素。动态是非常重要的，这就意味着XML元素在编译时是动态生成的而非静态。

以下实例动态定义了XML元素：

```
<%@page language="java" contentType="text/html"%>
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page">

<head><title>Generate XML Element</title></head>
<body>
<jsp:element name="xmlElement">
<jsp:attribute name="xmlElementAttr">
   Value for the attribute
</jsp:attribute>
<jsp:body>
   Body for XML element
</jsp:body>
</jsp:element>
</body>
</html>
```

执行时生成HTML代码如下：

```
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page">

<head><title>Generate XML Element</title></head>
<body>
<xmlElement xmlElementAttr="Value for the attribute">
   Body for XML element
</xmlElement>
</body>
</html>
```

## <jsp:text>动作元素

<jsp:text>动作元素允许在JSP页面和文档中使用写入文本的模板，语法格式如下：

```
<jsp:text>Template data</jsp:text>
```

以上文本模板不能包含其他元素，只能只能包含文本和EL表达式（注：EL表达式将在后续章节中介绍）。请注意，在XML文件中，您不能使用表达式如 ${whatever > 0}，因为>符号是非法的。 你可以使用 ${whatever gt 0}表达式或者嵌入在一个CDATA部分的值。

```
<jsp:text><![CDATA[<br>]]></jsp:text>
```

如果你需要在 XHTML 中声明 DOCTYPE,必须使用到<jsp:text>动作元素，实例如下：

```
<jsp:text><![CDATA[<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">]]>
</jsp:text>
<head><title>jsp:text action</title></head>
<body>

<books><book><jsp:text>
    Welcome to JSP Programming
</jsp:text></book></books>

</body>
</html>
```

你可以对以上实例尝试使用<jsp:text>及不使用该动作元素执行结果的区别。

</p jsp:setproperty动作有下面四个属性,如下表：<>VM1563:1 Resource interpreted as Image but transferred with MIME type text/html: "http://googleads.g.doubleclick.net/pagead/adview?ai=CYd_viH8rVOrQGoGt8gW1-o…dHQJkV0teBb6uOUXPCSK1MkoVQh3X4TKABpy_58Kajo6JdKAGIQ&sigh=pcHBvVQp9GM&vis=1". ads?client=ca-pub-5751451760833794&format=160x600&output=html&h=600&slotname=4106274865&adk=4689099…:1

# JSP 动作元素

与JSP指令元素不同的是，JSP动作元素在请求处理阶段起作用。JSP动作元素是用XML语法写成的。

利用JSP动作可以动态地插入文件、重用JavaBean组件、把用户重定向到另外的页面、为Java插件生成HTML代码。

动作元素只有一种语法，它符合XML标准：

```
<jsp:action_name attribute="value" />
```

动作元素基本上都是预定义的函数，JSP规范定义了一系列的标准动作，它用JSP作为前缀，可用的标准动作元素如下：

| 语法 | 描述 |
| --- | --- |
| jsp:include | 在页面被请求的时候引入一个文件。 |
| jsp:useBean | 寻找或者实例化一个JavaBean。 |
| jsp:setProperty | 设置JavaBean的属性。 |
| jsp:getProperty | 输出某个JavaBean的属性。 |
| jsp:forward | 把请求转到一个新的页面。 |
| jsp:plugin | 根据浏览器类型为Java插件生成OBJECT或EMBED标记。 |
| jsp:element | 定义动态XML元素 |
| jsp:attribute | 设置动态定义的XML元素属性。 |
| jsp:body | 设置动态定义的XML元素内容。 |
| jsp:text | 在JSP页面和文档中使用写入文本的模板 |

## 常见的属性

所有的动作要素都有两个属性：id属性和scope属性。

- **id**属性：

  id属性是动作元素的唯一标识，可以在JSP页面中引用。动作元素创建的id值可以通过PageContext来调用。

- **scope**属性：

该属性用于识别动作元素的生命周期。 id属性和scope属性有直接关系，scope属性定义了相关联id对象的寿命。 scope属性有四个可能的值： (a) page, (b)request, (c)session, 和 (d) application。

# <jsp:include>动作元素

<jsp:include>动作元素用来包含静态和动态的文件。该动作把指定文件插入正在生成的页面。语法格式如下：

```
<jsp:include page="relative URL" flush="true" />
```

前面已经介绍过include指令，它是在JSP文件被转换成Servlet的时候引入文件，而这里的jsp:include动作不同，插入文件的时间是在页面被请求的时候。

以下是include动作相关的属性列表。

| 属性 | 描述 |
| --- | --- |
| page | 包含在页面中的相对URL地址。 |
| flush | 布尔属性，定义在包含资源前是否刷新缓存区。 |

## 实例

以下我们定义了两个文件date.jsp和main.jsp，代码如下所示：

date.jsp文件代码：

```
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

main.jsp文件代码：

```
<html>
<head>
<title>The include Action Example</title>
</head>
<body>
<center>
<h2>The include action Example</h2>
<jsp:include page="date.jsp" flush="true" />
</center>
</body>
</html>
```

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

```
The include action Example
Today's date: 12-Sep-2013 14:54:22
```

## <jsp:useBean>动作元素

jsp:useBean动作用来装载一个将在JSP页面中使用的JavaBean。

这个功能非常有用，因为它使得我们既可以发挥Java组件重用的优势，同时也避免了损失JSP区别于Servlet的方便性。

jsp:useBean动作最简单的语法为：

```
<jsp:useBean id="name" class="package.class" />
```

在类载入后，我们既可以通过 jsp:setProperty 和 jsp:getProperty 动作来修改和检索bean的属性。

以下是useBean动作相关的属性列表。

| 属性 | 描述 |
|---|---|
| class | 指定Bean的完整包名。 |
| type | 指定将引用该对象变量的类型。 |
| beanName | 通过 java.beans.Beans 的 instantiate() 方法指定Bean的名字。 |

在给出具体实例前，让我们先来看下 jsp:setProperty 和 jsp:getProperty 动作元素：

## <jsp:setProperty>动作元素

jsp:setProperty用来设置已经实例化的Bean对象的属性，有两种用法。首先，你可以在jsp:useBean元素的外面（后面）使用jsp:setProperty，如下所示：

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName" property="someProperty" .../>
```

此时，不管jsp:useBean是找到了一个现有的Bean，还是新创建了一个Bean实例，jsp:setProperty都会执行。第二种用法是把jsp:setProperty放入jsp:useBean元素的内部，如下所示：

```
<jsp:useBean id="myName" ... >
...
   <jsp:setProperty name="myName" property="someProperty" .../>
</jsp:useBean>
```

此时，jsp:setProperty只有在新建Bean实例时才会执行，如果是使用现有实例则不执行jsp:setProperty。

jsp:setproperty动作有下面四个属性,如下表：

| 属性 | 描述 |
|---|---|
| name | name属性是必需的。它表示要设置属性的是哪个Bean。 |
| property | property属性是必需的。它表示要设置哪个属性。有一个特殊用法：如果property的值是"*"，表示所有名字和Bean属性名字匹配的请求参数都将被传递给相应的属性set方法。 |
| value | value 属性是可选的。该属性用来指定Bean属性的值。字符串数据会在目标类中通过标准的valueOf方法自动转换成数字、boolean、Boolean、 byte、Byte、char、Character。例如，boolean和Boolean类型的属性值（比如"true"）通过 Boolean.valueOf转换，int和Integer类型的属性值（比如"42"）通过Integer.valueOf转换。 value和param不能同时使用，但可以使用其中任意一个。 |
| param | param 是可选的。它指定用哪个请求参数作为Bean属性的值。如果当前请求没有参数，则什么事情也不做，系统不会把null传递给Bean属性的set方法。因此，你可以让Bean自己提供默认属性值，只有当请求参数明确指定了新值时才修改默认属性值。 |

## <jsp:getProperty>动作元素

jsp:getProperty动作提取指定Bean属性的值，转换成字符串，然后输出。语法格式如下：

```
<jsp:useBean id="myName" ... />
...
<jsp:getProperty name="myName" property="someProperty" .../>
```

下表是与getProperty相关联的属性：

| 属性 | 描述 |
|---|---|
| name | 要检索的Bean属性名称。Bean必须已定义。 |
| property | 表示要提取Bean属性的值 |

# 实例

以下实例我们使用了Bean:

```
/* 文件: TestBean.java */
package action;

public class TestBean {
   private String message = "No message specified";

   public String getMessage() {
      return(message);
   }
   public void setMessage(String message) {
      this.message = message;
   }
}
```

编译以上实例并生成 TestBean.class 文件，将该文件拷贝至服务器正式存放Java类的目录下，而不是保留给修改后能够自动装载的类的目录( 如：C:\apache-tomcat-7.0.2\webapps\WEB-INF\classes\action目录中，CLASSPATH 变量必须包含该路径。 )。例如，对于Java Web Server来说，Bean和所有Bean用到的类都应该放入classes目录，或者封装进jar文件后放入lib目录，但不应该放到servlets 下。
下面是一个很简单的例子，它的功能是装载一个Bean，然后设置/读取它的message属性。

现在让我们在main.jsp文件中调用该Bean:

```
 <html>
<head>
<title>Using JavaBeans in JSP</title>
</head>
<body>
<center>
<h2>Using JavaBeans in JSP</h2>

<jsp:useBean id="test" class="action.TestBean" />

<jsp:setProperty name="test"
                    property="message"
                    value="Hello JSP..." />

<p>Got message....</p>

<jsp:getProperty name="test" property="message" />

</center>
</body>
</html>
```

执行以上文件，输出如下所示：

```
Using JavaBeans in JSP
Got message....
Hello JSP...
```

# <jsp:forward> 动作元素

jsp:forward动作把请求转到另外的页面。jsp:forward标记只有一个属性page。语法格式如下所示：

```
<jsp:forward page="Relative URL" />
```

以下是forward相关联的属性：

| 属性 | 描述 |
| --- | --- |
| page | page属性包含的是一个相对URL。page的值既可以直接给出，也可以在请求的时候动态计算，可以是一个JSP页面或者一个 Java Servlet. |

## 实例

以下实例我们使用了两个文件，分别是： date.jps 和 main.jsp。

date.js文件代码如下：

```
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

main.jsp文件代码：

```
<html>
<head>
<title>The forward Action Example</title>
</head>
<body>
<center>
<h2>The forward action Example</h2>
<jsp:forward page="date.jsp" />
</center>
</body>
```

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

```
Today's date: 12-Sep-2010 14:54:22
```

## <jsp:plugin>动作元素

jsp:plugin动作用来根据浏览器的类型，插入通过Java插件 运行Java Applet所必需的OBJECT或EMBED元素。

如果需要的插件不存在，它会下载插件，然后执行Java组件。 Java组件可以是一个applet或一个JavaBean。

plugin动作有多个对应HTML元素的属性用于格式化Java 组件。param元素可用于向Applet 或 Bean 传递参数。

以下是使用plugin 动作元素的典型实例:

```
<jsp:plugin type="applet" codebase="dirname" code="MyApplet.class"
                           width="60" height="80">
   <jsp:param name="fontcolor" value="red" />
   <jsp:param name="background" value="black" />

   <jsp:fallback>
      Unable to initialize Java Plugin
   </jsp:fallback>

</jsp:plugin>
```

如果你有兴趣可以尝试使用applet来测试jsp:plugin动作元素，<fallback>元素是一个新元素，在组件出现故障的错误是发送给用户错误信息。

## <jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素

<jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素动态定义XML元素。动态是非常重要的，这就意味着XML元素在编译时是动态生成的而非静态。

以下实例动态定义了XML元素：

```
<%@page language="java" contentType="text/html"%>
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page">

<head><title>Generate XML Element</title></head>
<body>
<jsp:element name="xmlElement">
<jsp:attribute name="xmlElementAttr">
   Value for the attribute
</jsp:attribute>
<jsp:body>
   Body for XML element
</jsp:body>
</jsp:element>
</body>
</html>
```

执行时生成HTML代码如下：

```
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page">

<head><title>Generate XML Element</title></head>
<body>
<xmlElement xmlElementAttr="Value for the attribute">
   Body for XML element
</xmlElement>
</body>
</html>
```

## <jsp:text>动作元素

<jsp:text>动作元素允许在JSP页面和文档中使用写入文本的模板，语法格式如下：

```
<jsp:text>Template data</jsp:text>
```

以上文本模板不能包含其他元素，只能只能包含文本和EL表达式（注：EL表达式将在后续章节中介绍）。请注意，在XML文件中，您不能使用表达式如 ${whatever > 0}，因为>符号是非法的。 你可以使用 ${whatever gt 0}表达式或者嵌入在一个CDATA部分的值。

```
<jsp:text><![CDATA[<br>]]></jsp:text>
```

如果你需要在 XHTML 中声明 DOCTYPE,必须使用到<jsp:text>动作元素，实例如下：

```
<jsp:text><![CDATA[<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">]]>
</jsp:text>
<head><title>jsp:text action</title></head>
<body>

<books><book><jsp:text>
    Welcome to JSP Programming
</jsp:text></book></books>

</body>
</html>
```

你可以对以上实例尝试使用<jsp:text>及不使用该动作元素执行结果的区别。

# JSP 隐含对象

JSP隐含对象是JSP容器为每个页面提供的Java对象，开发者可以直接使用它们而不用显式声明。JSP隐含对象也被称为预定义变量。

JSP所支持的九大隐含对象：

| 对象 | 描述 |
| --- | --- |
| request | **HttpServletRequest**类的实例 |
| response | **HttpServletResponse**类的实例 |
| out | **PrintWriter**类的实例，用于把结果输出至网页上 |
| session | **HttpSession**类的实例 |
| application | **ServletContext**类的实例，与应用上下午有关 |
| config | **ServletConfig**类的实例 |
| pageContext | **PageContext**类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | **Exception**类的对象，代表发生错误的JSP页面中对应的异常对象 |

## request对象

request对象是javax.servlet.http.HttpServletRequest 类的实例。每当客户端请求一个JSP页面时，JSP引擎就会制造一个新的request对象来代表这个请求。

request对象提供了一系列方法来获取HTTP头信息，cookies，HTTP方法等等。

## response对象

response对象是javax.servlet.http.HttpServletResponse类的实例。当服务器创建request对象时会同时创建用于响应这个客户端的response对象。

response对象也定义了处理HTTP头模块的接口。通过这个对象，开发者们可以添加新的cookies，时间戳，HTTP状态码等等。

## out对象

out对象是 javax.servlet.jsp.JspWriter 类的实例，用来在response对象中写入内容。

最初的JspWriter类对象根据页面是否有缓存来进行不同的实例化操作。可以在page指令中使用buffered='false'属性来轻松关闭缓存。

JspWriter类包含了大部分java.io.PrintWriter类中的方法。不过，JspWriter新增了一些专为处理缓存而设计的方法。还有就是，JspWriter类会抛出IOExceptions异常，而PrintWriter不会。

下表列出了我们将会用来输出boolean，char，int，double，Srtring，object等类型数据的重要方法：

| 方法 | 描述 |
| --- | --- |
| **out.print(dataType dt)** | 输出Type类型的值 |
| **out.println(dataType dt)** | 输出Type类型的值然后换行 |
| **out.flush()** | 刷新输出流 |

# session对象

session对象是 javax.servlet.http.HttpSession 类的实例。和Java Servlets中的session对象有一样的行为。

session对象用来跟踪在各个客户端请求间的会话。

# application对象

application对象直接包装了servlet的ServletContext类的对象，是javax.servlet.ServletContext 类的实例。

这个对象在JSP页面的整个生命周期中都代表着这个JSP页面。这个对象在JSP页面初始化时被创建，随着jspDestroy()方法的调用而被移除。

通过向application中添加属性，则所有组成您web应用的JSP文件都能访问到这些属性。

# config对象

config对象是 javax.servlet.ServletConfig 类的实例，直接包装了servlet的ServletConfig类的对象。

这个对象允许开发者访问Servlet或者JSP引擎的初始化参数，比如文件路径等。

以下是config对象的使用方法，不是很重要，所以不常用：

```
config.getServletName();
```

它返回包含在<servlet-name>元素中的servlet名字，注意，<servlet-name>元素在WEB-INF\web.xml 文件中定义。

# pageContext 对象

pageContext对象是javax.servlet.jsp.PageContext 类的实例，用来代表整个JSP页面。

这个对象主要用来访问页面信息，同时过滤掉大部分实现细节。

这个对象存储了request对象和response对象的引用。application对象，config对象，session对象，out对象可以通过访问这个对象的属性来导出。

pageContext对象也包含了传给JSP页面的指令信息，包括缓存信息，ErrorPage URL,页面scope等。

PageContext类定义了一些字段，包括PAGE_SCOPE，REQUEST_SCOPE，SESSION_SCOPE， APPLICATION_SCOPE。它也提供了40余种方法，有一半继承自javax.servlet.jsp.JspContext 类。

其中一个重要的方法就是removeArribute()，它可接受一个或两个参数。比如，pageContext.removeArribute("attrName")移除四个scope中相关属性，但是下面这种方法只移除特定scope中的相关属性：

```
pageContext.removeAttribute("attrName", PAGE_SCOPE);
```

# page 对象

这个对象就是页面实例的引用。它可以被看做是整个JSP页面的代表。

page 对象就是this对象的同义词。

# exception 对象

exception 对象包装了从先前页面中抛出的异常信息。它通常被用来产生对出错条件的适当响应。

# JSP 客户端请求

当浏览器请求一个网页时，它会向网络服务器发送一系列不能被直接读取的信息，因为这些信息是作为HTTP信息头的一部分来传送的。您可以查阅HTTP协议来获得更多的信息。

下表列出了浏览器端信息头的一些重要内容，在以后的网络编程中将会经常见到这些信息：

| 信息 | 描述 |
| --- | --- |
| Accept | 指定浏览器或其他客户端可以处理的MIME类型。它的值通常为 **image/png** 或 **image/jpeg** |
| Accept-Charset | 指定浏览器要使用的字符集。比如 ISO-8859-1 |
| Accept-Encoding | 指定编码类型。它的值通常为 **gzip** 或**compress** |
| Accept-Language | 指定客户端首选语言，servlet会优先返回以当前语言构成的结果集，如果servlet支持这种语言的话。比如 en，en-us，ru等等 |
| Authorization | 在访问受密码保护的网页时识别不同的用户 |
| Connection | 表明客户端是否可以处理HTTP持久连接。持久连接允许客户端或浏览器在一个请求中获取多个文件。**Keep-Alive** 表示启用持久连接 |
| Content-Length | 仅适用于POST请求，表示 POST 数据的字节数 |
| Cookie | 返回先前发送给浏览器的cookies至服务器 |
| Host | 指出原始URL中的主机名和端口号 |
| If-Modified-Since | 表明只有当网页在指定的日期被修改后客户端才需要这个网页。 服务器发送304码给客户端，表示没有更新的资源 |
| If-Unmodified-Since | 与If-Modified-Since相反， 只有文档在指定日期后仍未被修改过，操作才会成功 |
| Referer | 标志着所引用页面的URL。比如，如果你在页面1，然后点了个链接至页面2，那么页面1的URL就会包含在浏览器请求页面2的信息头中 |
| User-Agent | 用来区分不同浏览器或客户端发送的请求，并对不同类型的浏览器返回不同的内容 |

# HttpServletRequest类

request对象是javax.servlet.http.HttpServletRequest类的实例。每当客户端请求一个页面时，JSP引擎就会产生一个新的对象来代表这个请求。

request对象提供了一系列方法来获取HTTP信息头，包括表单数据，cookies，HTTP方法等等。

接下来将会介绍一些在JSP编程中常用的获取HTTP信息头的方法。详细内容请见下表：

| 方法 | 描述 |
|---|---|
| **Cookie[] getCookies()** | 返回客户端所有的Cookie的数组 |
| **Enumeration getAttributeNames()** | 返回request对象的所有属性名称的集合 |
| **Enumeration getHeaderNames()** | 返回所有HTTP头的名称集合 |
| **Enumeration getParameterNames()** | 返回请求中所有参数的集合 |
| **HttpSession getSession()** | 返回request对应的session对象，如果没有，则创建一个 |
| **HttpSession getSession(boolean create)** | 返回request对应的session对象，如果没有并且参数create为true，则返回一个新的session对象 |
| **Locale getLocale()** | 返回当前页的Locale对象，可以在response中设置 |
| **Object getAttribute(String name)** | 返回名称为name的属性值，如果不存在则返回null。 |
| **ServletInputStream getInputStream()** | 返回请求的输入流 |
| **String getAuthType()** | 返回认证方案的名称，用来保护servlet，比如 "BASIC" 或者 "SSL" 或 null 如果 JSP没设置保护措施 |
| **String getCharacterEncoding()** | 返回request的字符编码集名称 |
| **String getContentType()** | 返回request主体的MIME类型，若未知则返回null |
| **String getContextPath()** | 返回request URI中指明的上下文路径 |
| **String getHeader(String** | |

| | |
|---|---|
| **name)** | |
| **String getMethod()** | 返回此request中的HTTP方法，比如 GET,，POST，或PUT |
| **String getParameter(String name)** | 返回此request中name指定的参数，若不存在则返回null |
| **String getPathInfo()** | 返回任何额外的与此request URL相关的路径 |
| **String getProtocol()** | 返回此request所使用的协议名和版本 |
| **String getQueryString()** | 返回此 request URL包含的查询字符串 |
| **String getRemoteAddr()** | 返回客户端的IP地址 |
| **String getRemoteHost()** | 返回客户端的完整名称 |
| **String getRemoteUser()** | 返回客户端通过登录认证的用户，若用户未认证则返回null |
| **String getRequestURI()** | 返回request的URI |
| **String getRequestedSessionId()** | 返回request指定的session ID |
| **String getServletPath()** | 返回所请求的servlet路径 |
| **String[] getParameterValues(String name)** | 返回指定名称的参数的所有值，若不存在则返回null |
| **boolean isSecure()** | 返回request是否使用了加密通道，比如HTTPS |
| **int getContentLength()** | 返回request主体所包含的字节数，若未知的返回-1 |
| **int getIntHeader(String name)** | 返回指定名称的request信息头的值 |
| **int getServerPort()** | 返回服务器端口号 |

# HTTP信息头示例

在这个例子中，我们会使用HttpServletRequest类的getHeaderNames()方法来读取HTTP信息头。这个方法以枚举的形式返回当前HTTP请求的头信息。

获取Enumeration对象后，用标准的方式来遍历Enumeration对象，用hasMoreElements()方法来确定什么时候停止，用nextElement()方法来获得每个参数的名字。

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>HTTP Header Request Example</title>
</head>
<body>
<center>
<h2>HTTP Header Request Example</h2>
<table width="100%" border="1" align="center">
<tr bgcolor="#949494">
<th>Header Name</th><th>Header Value(s)</th>
</tr>
<%
   Enumeration headerNames = request.getHeaderNames();
   while(headerNames.hasMoreElements()) {
      String paramName = (String)headerNames.nextElement();
      out.print("<tr><td>" + paramName + "</td>\n");
      String paramValue = request.getHeader(paramName);
      out.println("<td> " + paramValue + "</td></tr>\n");
   }
%>
</table>
</center>
</body>
</html>
```

访问main.jsp，将会得到以下结果：

```
<h1>HTTP Header Request Example</h1>

<table width="100%" border="1" align="center">

<tbody>

<tr><th>Header Name</th><th>Header Value(s)</th></tr>

<tr><td>accept</td><td>*/*</td></tr>

<tr><td>accept-language</td><td>en-us</td></tr>

<tr><td>user-agent</td><td>Mozilla/4.0 (compatible; MSIE 7.0; Windo

<tr><td>accept-encoding</td><td>gzip, deflate</td></tr>

<tr><td>host</td><td>localhost:8080</td></tr>

<tr><td>connection</td><td>Keep-Alive</td></tr>

<tr><td>cache-control</td><td>no-cache</td></tr>

</tbody>

</table>
```

您可以在上面代码中尝试HttpServletRequest类的其它方法。

# JSP 服务器响应

Response响应对象主要将JSP容器处理后的结果传回到客户端。可以通过response变量设置HTTP的状态和向客户端发送数据，如Cookie、HTTP文件头信息等。

一个典型的响应看起来就像下面这样：

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```

状态行包含HTTP版本信息，比如HTTP/1.1，一个状态码，比如200，还有一个非常短的信息对应着状态码，比如OK。

下表摘要出了HTTP1.1响应头中最有用的部分，在网络编程中您将会经常见到它们：

| 响应头 | 描述 |
| --- | --- |
| Allow | 指定服务器支持的request方法（GET，POST等等） |
| Cache-Control | 指定响应文档能够被安全缓存的情况。通常取值为 **public****，****private** 或**no-cache** 等等。 Public意味着文档可缓存，Private意味着文档只为单用户服务并且只能使用私有缓存。No-cache 意味着文档不被缓存。 |
| Connection | 命令浏览器是否要使用持久的HTTP连接。**close****值** 命令浏览器不使用持久HTTP连接，而keep-alive 意味着使用持久化连接。 |
| Content-Disposition | 让浏览器要求用户将响应以给定的名称存储在磁盘中 |
| Content-Encoding | 指定传输时页面的编码规则 |
| Content-Language | 表述文档所使用的语言，比如en， en-us,，ru等等 |
| Content-Length | 表明响应的字节数。只有在浏览器使用持久化 (keep-alive) HTTP 连接时才有用 |
| Content-Type | 表明文档使用的MIME类型 |
| Expires | 指明啥时候过期并从缓存中移除 |
| Last-Modified | 指明文档最后修改时间。客户端可以 缓存文档并且在后续的请求中提供一个 **If-Modified-Since**请求头 |
| Location | 在300秒内，包含所有的有一个状态码的响应地址，浏览器会自动重连然后检索新文档 |
| Refresh | 指明浏览器每隔多久请求更新一次页面。 |
| Retry-After | 与503 (Service Unavailable)一起使用来告诉用户多久后请求将会得到响应 |
| Set-Cookie | 指明当前页面对应的cookie |

## HttpServletResponse类

response对象是javax.servlet.http.HttpServletRequest类的一个实例。就像服务器会创建request对象一样，它也会创建一个客户端响应。

response对象定义了处理创建HTTP信息头的接口。通过使用这个对象，开发者们可以添加新的cookie或时间戳，还有HTTP状态吗等等。

下表列出了用来设置HTTP响应头的方法，这些方法由HttpServletResponse 类提供：

| 方法 | 描述 |
|---|---|
| **String encodeRedirectURL(String url)** | 对sendRedirect()方法使用的URL进行编码 |
| **String encodeURL(String url)** | 将URL编码，回传包含Session ID的URL |
| **boolean containsHeader(String name)** | 返回指定的响应头是否存在 |
| **boolean isCommitted()** | 返回响应是否已经提交到客户端 |
| **void addCookie(Cookie cookie)** | 添加指定的cookie至响应中 |
| **void addDateHeader(String name, long date)** | 添加指定名称的响应头和日期值 |
| **void addHeader(String name, String value)** | 添加指定名称的响应头和值 |
| **void addIntHeader(String name, int value)** | 添加指定名称的响应头和int值 |
| **void flushBuffer()** | 将任何缓存中的内容写入客户端 |
| **void reset()** | 清除任何缓存中的任何数据，包括状态码和各种响应头 |
| **void resetBuffer()** | 清除基本的缓存数据，不包括响应头和状态码 |
| **void sendError(int sc)** | 使用指定的状态码向客户端发送一个出错响应，然后清除缓存 |
| **void sendError(int sc, String msg)** | 使用指定的状态码和消息向客户端发送一个出错响应 |
| **void sendRedirect(String location)** | 使用指定的URL向客户端发送一个临时的间接响应 |
| **void setBufferSize(int size)** | 设置响应体的缓存区大小 |
| **void setCharacterEncoding(String charset)** | 指定响应的编码集（MIME字符集），例如UTF-8 |
| **void setContentLength(int len)** | 指定HTTP servlets中响应的内容的长度，此方法用来设置 HTTP Content-Length 信息头 |

| | |
|---|---|
| **void setContentType(String type)** | 设置响应的内容的类型，如果响应还未被提交的话 |
| **void setDateHeader(String name, long date)** | 使用指定名称和值设置响应头的名称和内容 |
| **void setHeader(String name, String value)** | 使用指定名称和值设置响应头的名称和内容 |
| **void setIntHeader(String name, int value)** | 使用指定名称和值设置响应头的名称和内容 |
| **void setLocale(Locale loc)** | 设置响应的语言环境，如果响应尚未被提交的话 |
| **void setStatus(int sc)** | 设置响应的状态码 |

# HTTP响应头程序示例

接下来的例子使用setIntHeader()方法和setRefreshHeader()方法来模拟一个数字时钟：

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>Auto Refresh Header Example</title>
</head>
<body>
<center>
<h2>Auto Refresh Header Example</h2>
<%
   // 设置每隔5秒自动刷新
   response.setIntHeader("Refresh", 5);
   // 获取当前时间
   Calendar calendar = new GregorianCalendar();
   String am_pm;
   int hour = calendar.get(Calendar.HOUR);
   int minute = calendar.get(Calendar.MINUTE);
   int second = calendar.get(Calendar.SECOND);
   if(calendar.get(Calendar.AM_PM) == 0)
      am_pm = "AM";
   else
      am_pm = "PM";
   String CT = hour+":"+ minute +":"+ second +" "+ am_pm;
   out.println("Current Time is: " + CT + "\n");
%>
</center>
</body>
</html>
```

将以上代码保存为main.jsp，然后通过浏览器访问它。它将会每隔5秒显示一下系统当前时间。

运行结果如下：

```
Auto Refresh Header Example
Current Time is: 9:44:50 PM
```

您也可以自己动手修改以上代码，试试使用其他的方法，将能得到更深的体会。

# JSP HTTP 状态码

HTTP请求与HTTP响应的格式相近，都有着如下结构：

- 以状态行+CRLF（回车换行）开始
- 零行或多行头模块+CRLF
- 一个空行，比如CRLF
- 可选的消息体比如文件，查询数据，查询输出

举例来说，一个服务器响应头看起来就像下面这样：

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```

状态行包含HTTP版本，一个状态码，和状态码相对应的短消息。

下表列出了可能会从服务器返回的HTTP状态码和与之关联的消息：

| 状态码 | 消息 | 描述 |
|---|---|---|
| 100 | Continue | 只有一部分请求被服务器接收，但只要没被服务器拒绝，客户端就会延续这个请求 |
| 101 | Switching Protocols | 服务器交换机协议 |
| 200 | OK | 请求被确认 |
| 201 | Created | 请求时完整的，新的资源被创建 |
| 202 | Accepted | 请求被接受，但未处理完 |
| 203 | Non-authoritative Information | |

| | | |
|---|---|---|
| 204 | No Content | |
| 205 | Reset Content | |
| 206 | Partial Content | |
| 300 | Multiple Choices | 一个超链接表，用户可以选择一个超链接并访问，最大支持5个超链接 |
| 301 | Moved Permanently | 被请求的页面已经移动到了新的URL下 |
| 302 | Found | 被请求的页面暂时性地移动到了新的URL下 |
| 303 | See Other | 被请求的页面可以在一个不同的URL下找到 |
| 304 | Not Modified | |
| 305 | Use Proxy | |
| 306 | *Unused* | 已经不再使用此状态码，但状态码被保留 |
| 307 | Temporary Redirect | 被请求的页面暂时性地移动到了新的URL下 |
| 400 | Bad Request | 服务器无法识别请求 |
| 401 | Unauthorized | 被请求的页面需要用户名和密码 |
| 402 | Payment Required | 目前还不能使用此状态码 |
| 403 | Forbidden | 禁止访问所请求的页面 |
| 404 | Not Found | 服务器无法找到所请求的页面 |
| 405 | Method Not Allowed | 请求中所指定的方法不被允许 |
| 406 | Not Acceptable | 服务器只能创建一个客户端无法接受的响应 |
| 407 | Proxy Authentication Required | 在请求被服务前必须认证一个代理服务器 |
| 408 | Request Timeout | 请求时间超过了服务器所能等待的时间，连接被断开 |
| 409 | Conflict | 请求有矛盾的地方 |
| 410 | Gone | 被请求的页面不再可用 |
| 411 | Length Required | "Content-Length"没有被定义，服务器拒绝接受请求 |

| | | |
|---|---|---|
| 412 | Precondition Failed | 请求的前提条件被服务器评估为false |
| 413 | Request Entity Too Large | 因为请求的实体太大，服务器拒绝接受请求 |
| 414 | Request-url Too Long | 服务器拒绝接受请求，因为URL太长。多出现在把"POST"请求转换为"GET"请求时所附带的大量查询信息 |
| 415 | Unsupported Media Type | 服务器拒绝接受请求，因为媒体类型不被支持 |
| 417 | Expectation Failed | |
| 500 | Internal Server Error | 请求不完整，服务器遇见了出乎意料的状况 |
| 501 | Not Implemented | 请求不完整，服务器不提供所需要的功能 |
| 502 | Bad Gateway | 请求不完整，服务器从上游服务器接受了一个无效的响应 |
| 503 | Service Unavailable | 请求不完整，服务器暂时重启或关闭 |
| 504 | Gateway Timeout | 网关超时 |
| 505 | HTTP Version Not Supported | 服务器不支持所指定的HTTP版本 |

# 设置HTTP状态码的方法

下表列出了HttpServletResponse 类中用来设置状态码的方法：

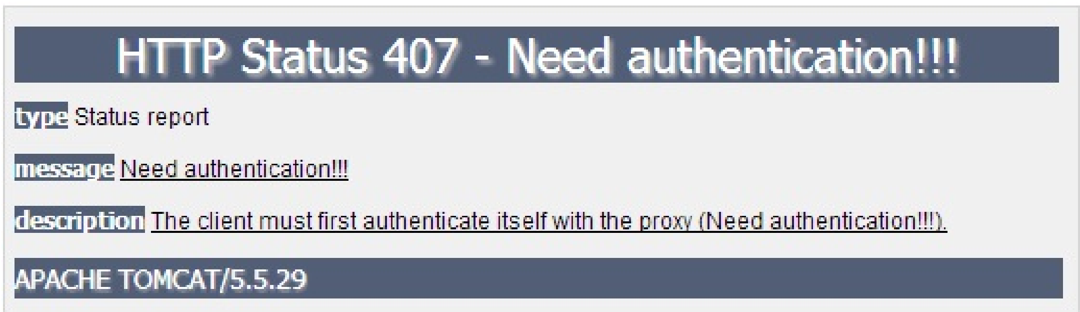
| 方法 | 描述 |
| --- | --- |
| **public void setStatus ( int statusCode )** | 此方法可以设置任意的状态码。如果您的响应包含一个特殊的状态码和一个文档，请确保在用PrintWriter返回任何内容前调用setStatus方法 |
| **public void sendRedirect(String url)** | 此方法产生302响应，同时产生一个 *Location* 头告诉URL 一个新的文档 |
| **public void sendError(int code, String message)** | 此方法将一个状态码(通常为 404)和一个短消息，自动插入HTML文档中并发回给客户端 |

# HTTP状态码程序示例

接下来的例子将会发送407错误码给浏览器，然后浏览器将会告诉您"Need authentication!!!"。

```
<html>
<head>
<title>Setting HTTP Status Code</title>
</head>
<body>
<%
   // 设置错误代码，并说明原因
   response.sendError(407, "Need authentication!!!" );
%>
</body>
</html>
```

访问以上JSP页面，将会得到以下结果：




您也可以试试使用其他的状态码，看会不会得到什么意想不到结果。

# JSP 表单处理

我们在浏览网页的时候，经常需要向服务器提交信息，并让后台程序处理。浏览器中使用 GET 和 POST 方法向服务器提交数据。

## GET 方法

GET方法将请求的编码信息添加在网址后面，网址与编码信息通过"?"号分隔。如下所示：

```
http://www.w3cschool.cc/hello?key1=value1&key2=value2
```

GET方法是浏览器默认传递参数的方法，一些敏感信息，如密码等建议不使用GET方法。

用get时，传输数据的大小有限制 （注意不是参数的个数有限制），最大为1024字节。

## POST 方法

一些敏感信息，如密码等我们可以同过POST方法传递，post提交数据是隐式的。

POST提交数据是不可见的，GET是通过在url里面传递的（可以看一下你浏览器的地址栏）。

JSP使用getParameter()来获得传递的参数，getInputStream()方法用来处理客户端的二进制数据流的请求。

## JSP 读取表单数据

- **getParameter():** 使用 request.getParameter() 方法来获取表单参数的值。
- **getParameterValues():** 获得如checkbox类（名字相同，但值有多个）的数据。 接收数组变量 ，如checkobx类型
- **getParameterNames():**该方法可以取得所有变量的名称，该方法返回一个Emumeration。
- **getInputStream():**调用此方法来读取来自客户端的二进制数据流。

## 使用URL的 GET 方法实例

以下是一个简单的URL,并使用GET方法来传递URL中的参数：

```
http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI
```

以下是main.jsp文件的JSP程序用于处理客户端提交的表单数据，我们使用getParameter()方法来获取提交的数据：

```
<html>
<head>
<title>Using GET Method to Read Form Data</title>
</head>
<body>
<center>
<h1>Using GET Method to Read Form Data</h1>
<ul>
<li><p><b>First Name:</b>
   <%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last  Name:</b>
   <%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>
```

接下来我们通过浏览器访问*http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI* 输出结果如下所示：

```
Using GET Method to Read Form Data
First Name: ZARA

Last Name: ALI
```

# 使用表单的 GET 方法实例

以下是一个简单的HTML表单，该表单通过GET方法将客户端数据提交 到main.jsp文件中：

```
<html>
<body>
<form action="main.jsp" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

将以上HTML代码保存到Hello.htm文件中。 将该文件放置于<Tomcat安装目录>/webapps/ROOT 目录下。 通过访问 *http://localhost:8080/Hello.htm*， 输出界面如下所示：

First Name:

Last Name: Submit

在"First Name" 与 "Last Name"两个表单中填入信息，并点击"Submit"按钮，它将输出结果。

## 使用表单的 POST 方法实例

接下来让我们使用POST方法来传递表单数据，修改main.jsp与Hello.htm文件代码，如下所示：

main.jsp文件代码：

```
<html>
<head>
<title>Using GET and POST Method to Read Form Data</title>
</head>
<body>
<center>
<h1>Using GET Method to Read Form Data</h1>
<ul>
<li><p><b>First Name:</b>
   <%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last  Name:</b>
   <%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>
```

以下是Hello.htm修改后的代码：

```
<html>
<body>
<form action="main.jsp" method="POST">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

通过浏览器访问 *http://localhost:8080/Hello.htm*，输出如下：

First Name:

Last Name: Submit

在"First Name" 与 "Last Name"两个表单中填入信息，并点击"Submit"按钮，它将输出结果。

# 传递 Checkbox 数据到JSP程序

复选框 checkbox 可以传递一个甚至多个数据。

以下是一个简单的HTML代码，并将代码保存在CheckBox.htm文件中：

```
<html>
<body>
<form action="main.jsp" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics"  /> Physics
<input type="checkbox" name="chemistry" checked="checked" />
                                                Chemistry
<input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

以上代码在浏览器访问如下所示：

以下为main.jsp文件代码，用于处理复选框数据：

```
<html>
<head>
<title>Reading Checkbox Data</title>
</head>
<body>
<center>
<h1>Reading Checkbox Data</h1>
<ul>
<li><p><b>Maths Flag:</b>
   <%= request.getParameter("maths")%>
</p></li>
<li><p><b>Physics Flag:</b>
   <%= request.getParameter("physics")%>
</p></li>
<li><p><b>Chemistry Flag:</b>
   <%= request.getParameter("chemistry")%>
</p></li>
</ul>
</body>
</html>
```

以上实例输出结果为：

Maths Physics Chemistry Select Subject

# 读取所有表单参数

以下我们将使用 HttpServletRequest 的getParameterNames()来读取所有可用的表单参数,该方法可以取得所有变量的名称，该方法返回一个Emumeration。

一旦我们有了一个Enumeration（枚举），我们就可以调用hasMoreElements（）方法来确定何时停止使用和nextElement（）方法来获得每个参数的名称。

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>HTTP Header Request Example</title>
</head>
<body>
<center>
<h2>HTTP Header Request Example</h2>
<table width="100%" border="1" align="center">
<tr bgcolor="#949494">
<th>Param Name</th><th>Param Value(s)</th>
</tr>
<%
   Enumeration paramNames = request.getParameterNames();

   while(paramNames.hasMoreElements()) {
      String paramName = (String)paramNames.nextElement();
      out.print("<tr><td>" + paramName + "</td>\n");
      String paramValue = request.getHeader(paramName);
      out.println("<td> " + paramValue + "</td></tr>\n");
   }
%>
</table>
</center>
</body>
</html>
```

以下是Hello.htm文件的内容:

```
<html>
<body>
<form action="main.jsp" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics"  /> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

现在我们通过浏览器访问 Hello.htm 文件并提交数据，输出结果如下：

## Reading All Form Parameters

| Param Name | Param Value(s) |
|---|---|
| maths | on |
| chemistry | on |

你可以尝试使用以上的JSP代码读取其它对象，如文本框，单选按钮或下拉框等等其他形式的数据。

# JSP 过滤器

Servlet和JSP中的过滤器都是Java类，它们存在的目的如下：

- 在请求访问后端资源时拦截它
- 管理从服务器返回给客户端的响应

下面列出了多种常用的过滤器类型：

- 认证过滤器
- 数据压缩过滤器
- 加密过滤器
- 触发资源访问事件的过滤器
- 图像转换过滤器
- 登录和验证过滤器
- MIME类型链过滤器
- 令牌过滤器
- 转换XML内容的XSL/T过滤器

过滤器将会被插入进web.xml文件中，然后映射servlet、JSP文件的名字，或URL模式。部署描述文件web.xml可以在 <Tomcat-installation-directory>\conf 目录下找到。

当JSP容器启动网络应用程序时，它会创建每一个过滤器的实例，这些过滤器必须在部署描述文件web.xml中声明，并且按声明的顺序执行。

## Servlet过滤器方法

一个过滤器就是一个Java类，它实现了javax.servlet.Filter 接口。javax.servlet.Filter接口定义了三个方法：

| 方法 | 描述 |
|---|---|
| **public void doFilter (ServletRequest, ServletResponse, FilterChain)** | 该方法在每次一个请求/响应对因客户端在链的末端请求资源而通过链传递时由容器调用。 |
| **public void init(FilterConfig filterConfig)** | 该方法由 Web 容器调用，指示一个过滤器被放入服务。 |
| **public void destroy()** | 该方法由 Web 容器调用，指示一个过滤器被取出服务。 |

## JSP过滤器示例

这个例子将会打印IP地址和每次访问JSP文件的日期时间。当然，这只是个简单的例子，让您了解一些简单的过滤器用法，但是可以使用这些概念来自行构造更复杂的程序。

```
//  引入Java包
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 实现 Filter 类
public class LogFilter implements Filter  {
   public void  init(FilterConfig config)
                         throws ServletException{
      // 获取初始化参数
      String testParam = config.getInitParameter("test-param");

      //打印初始化参数
      System.out.println("Test Param: " + testParam);
   }
   public void  doFilter(ServletRequest request,
                 ServletResponse response,
                 FilterChain chain)
                 throws java.io.IOException, ServletException {

      // 获取客户端ip地址
      String ipAddress = request.getRemoteAddr();

      // 输出ip地址及当前时间
      System.out.println("IP "+ ipAddress + ", Time "
                                 + new Date().toString());

      // 传递请求道过滤器链
      chain.doFilter(request,response);
   }
   public void destroy( ){
      /* 在Filter实例在服务器上被移除前调用。*/
   }
}
```

编译LogFilter.java文件，然后将编译后的class文件放在<Tomcat安装目录>/webapps/ROOT/WEB-INF/classes目录下。

## web.xml文件中的JSP过滤器映射

过滤器被定义，然后映射成一个URL或JSP文件名，与servlet被定义然后映射的方式差不多。在部署描述文件web.xml中，使用<filter>标签来进行过滤器映射：

```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
   <init-param>
      <param-name>test-param</param-name>
      <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

上述过滤器将会应用在所有servlet和JSP程序中，因为我们在配置中指定了" /*"。您也可以指定一个servlet或JSP路径，如果您只想要将过滤器应用在少数几个servlet或JSP程序中的话。

现在，像平常一样访问servlet或JSP页面，您就会发现服务器日志中产生了关于此次访问的记录。您也可以使用Log4J记录器来把日志记录在其它文件中。

## 使用多重过滤器

您的网络应用程序可以定义很多不同的过滤器。现在，您定义了两个过滤器，AuthenFilter和LogFilter，其它的步骤与前面讲的一样，除非要创建一个不同的映射，就像下面这样：

```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
   <init-param>
      <param-name>test-param</param-name>
      <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>

<filter>
   <filter-name>AuthenFilter</filter-name>
   <filter-class>AuthenFilter</filter-class>
   <init-param>
      <param-name>test-param</param-name>
      <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>

<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>AuthenFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

## 过滤器的应用顺序

在web.xml中<filter>元素的映射顺序决定了容器应用这些过滤器的顺序。要反转应用的顺序，您只需要反转web.xml中<filter>元素的定义顺序就行了。

比如，上面的例子会首先应用 LogFilter然后再应用AuthenFilter，但是下面这个例子将会反转应用的顺序：

```
<filter-mapping>
   <filter-name>AuthenFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

# JSP Cookies 处理

Cookies是存储在客户机的文本文件，它们保存了大量轨迹信息。在servlet技术基础上，JSP显然能够提供对HTTP cookies的支持。

通常有三个步骤来识别回头客：

- 服务器脚本发送一系列cookies至浏览器。比如名字，年龄，ID号码等等。
- 浏览器在本地机中存储这些信息，以备不时之需。
- 当下一次浏览器发送任何请求至服务器时，它会同时将这些cookies信息发送给服务器，然后服务器使用这些信息来识别用户或者干些其它事情。

本章节将会传授您如何去设置或重设cookie的方法，还有如何访问它们及如何删除它们。

## Cookie剖析

Cookies通常在HTTP信息头中设置（虽然JavaScript能够直接在浏览器中设置cookies）。在JSP中，设置一个cookie需要发送如下的信息头给服务器：

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
                 path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

正如您所见，Set-Cookie信息头包含一个键值对，一个GMT（格林尼治标准）时间，一个路径，一个域名。键值对会被编码为URL。有效期域是个指令，告诉浏览器在什么时候之后就可以清除这个cookie。

如果浏览器被配置成可存储cookies，那么它将会保存这些信息直到过期。如果用户访问的任何页面匹配了cookie中的路径和域名，那么浏览器将会重新将这个cookie发回给服务器。浏览器端的信息头长得就像下面这样：

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

JSP脚本通过request对象中的getCookies()方法来访问这些cookies，这个方法会返回一个Cookie对象的数组。

## Servlet Cookies 方法

下表列出了Cookie对象中常用的方法：

| 方法 | 描述 |
|---|---|
| **public void setDomain(String pattern)** | 设置cookie的域名，比如w3cschool.cc |
| **public String getDomain()** | 获取cookie的域名，比如w3cschool.cc |
| **public void setMaxAge(int expiry)** | 设置cookie有效期，以秒为单位，默认有效期为当前session的存活时间 |
| **public int getMaxAge()** | 获取cookie有效期，以秒为单位，默认为-1，表明cookie会活到浏览器关闭为止 |
| **public String getName()** | 返回 cookie的名称，名称创建后将不能被修改 |
| **public void setValue(String newValue)** | 设置 cookie的值 |
| **public String getValue()** | 获取cookie的值 |
| **public void setPath(String uri)** | 设置cookie 的路径，默认为当前页面目录下的所有URL，还有此目录下的所有子目录 |
| **public String getPath()** | 获取cookie 的路径 |
| **public void setSecure(boolean flag)** | 指明cookie是否要加密传输 |
| **public void setComment(String purpose)** | 设置注释描述 cookie的目的。当浏览器将cookie展现给用户时，注释将会变得非常有用 |
| **public String getComment()** | 返回描述cookie目的的注释，若没有则返回null |

## 使用JSP设置Cookies

使用JSP设置cookie包含三个步骤：

**(1)**创建一个**Cookie**对象： 调用Cookie的构造函数，使用一个cookie名称和值做参数，它们都是字符串。

```
Cookie cookie = new Cookie("key","value");
```

请务必牢记，名称和值中都不能包含空格或者如下的字符：

```
[ ] ( ) = , " / ? @ : ;
```

**(2)** 设置有效期：调用setMaxAge()函数表明cookie在多长时间（以秒为单位）内有效。下面的操作将有效期设为了24小时。

```
cookie.setMaxAge(60*60*24);
```

**(3)** 将**cookie**发送至**HTTP**响应头中：调用response.addCookie()函数来向HTTP响应头中添加cookies。

```
response.addCookie(cookie);
```

## 实例演示

```
<%
   // 为 first_name 和 last_name设置cookie
   Cookie firstName = new Cookie("first_name",
               request.getParameter("first_name"));
   Cookie lastName = new Cookie("last_name",
              request.getParameter("last_name"));

   // 设置cookie过期时间为24小时。
   firstName.setMaxAge(60*60*24);
   lastName.setMaxAge(60*60*24);

   // 在响应头部添加cookie
   response.addCookie( firstName );
   response.addCookie( lastName );
%>
<html>
<head>
<title>Setting Cookies</title>
</head>
<body>
<center>
<h1>Setting Cookies</h1>
</center>
<ul>
<li><p><b>First Name:</b>
   <%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last  Name:</b>
   <%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>
```

将上面两个文件放在<Tomcat安装目录>/webapps/ROOT目录下，然后访问http://localhost:8080/hello.jsp，将会得到如下输出结果：

First Name:

Last Name: Submit

试着输入First Name和Last Name，然后点击提交按钮，它将会在您的屏幕中显示first name和last name，并且设置first name和last name两个cookie，下一次点击提交按钮时会发给服务器。

# 使用JSP读取Cookies

想要读取cookies，您就需要调用request.getCookies()方法来获得一个javax.servlet.http.Cookie对象的数组，然后遍历这个数组，使用getName()方法和getValue()方法来获取每一个cookie的名称和值。

让我们来读取上个例子中的cookies。

```
<html>
<head>
<title>Reading Cookies</title>
</head>
<body>
<center>
<h1>Reading Cookies</h1>
</center>
<%
   Cookie cookie = null;
   Cookie[] cookies = null;
   // 获取cookies的数据,是一个数组
   cookies = request.getCookies();
   if( cookies != null ){
      out.println("<h2> Found Cookies Name and Value</h2>");
      for (int i = 0; i < cookies.length; i++){
         cookie = cookies[i];
         out.print("Name : " + cookie.getName( ) + ",  ");
         out.print("Value: " + cookie.getValue( )+" <br/>");
      }
  }else{
      out.println("<h2>No cookies founds</h2>");
  }
%>
</body>
</html>
```

如果您把first name cookie设置成"John"，last name设置成"Player"，访问http://localhost:8080/main.jsp，将会得到如下输出结果：

```
Found Cookies Name and Value
Name : first_name, Value: John
Name : last_name, Value: Player
```

# 使用JSP删除Cookies

删除cookies非常简单。如果您想要删除一个cookie，按照下面给的步骤来做就行了：

- 获取一个已经存在的cookie然后存储在Cookie对象中。
- 将cookie的有效期设置为0。
- 将这个cookie重新添加进响应头中。

## 实例演示

下面的程序删除一个名为"first_name"的cookie，当您下次运行main.jsp时，first_name将会为null。

```
<html>
<head>
<title>Reading Cookies</title>
</head>
<body>
<center>
<h1>Reading Cookies</h1>
</center>
<%
   Cookie cookie = null;
   Cookie[] cookies = null;
   // 获取当前域名下的cookies，是一个数组
   cookies = request.getCookies();
   if( cookies != null ){
      out.println("<h2> Found Cookies Name and Value</h2>");
      for (int i = 0; i < cookies.length; i++){
         cookie = cookies[i];
         if((cookie.getName( )).compareTo("first_name") == 0 ){
            cookie.setMaxAge(0);
            response.addCookie(cookie);
            out.print("Deleted cookie: " +
            cookie.getName( ) + "<br/>");
         }
         out.print("Name : " + cookie.getName( ) + ",  ");
         out.print("Value: " + cookie.getValue( )+" <br/>");
      }
  }else{
      out.println(
      "<h2>No cookies founds</h2>");
  }
%>
</body>
</html>
```

访问它，将会得到如下输出结果：

```
Cookies Name and Value
Deleted cookie : first_name
Name : first_name, Value: John
Name : last_name, Value: Player
```

再次访问http://localhost:8080/main.jsp，将会得到如下结果：

```
Found Cookies Name and Value
Name : last_name, Value: Player
```

您也可以手动在浏览器中删除cookies。点击Tools菜单项，然后选择Internet Options，点击Delete Cookies，就能删除所有cookies了。

# JSP Session

HTTP是无状态协议，这意味着每次客户端检索网页时，都要单独打开一个服务器连接，因此服务器不会记录下先前客户端请求的任何信息。

有三种方法来维持客户端与服务器的会话：

## Cookies

网络服务器可以指定一个唯一的session ID作为cookie来代表每个客户端，用来识别这个客户端接下来的请求。

这可能不是一种有效的方式，因为很多时候浏览器并不一定支持cookie，所以我们不建议使用这种方法来维持会话。

## 隐藏表单域

一个网络服务器可以发送一个隐藏的HTML表单域和一个唯一的session ID，就像下面这样：

```
<input type="hidden" name="sessionid" value="12345">
```

这个条目意味着，当表单被提交时，指定的名称和值将会自动包含在GET或POST数据中。每当浏览器发送一个请求，session_id的值就可以用来保存不同浏览器的轨迹。

这种方式可能是一种有效的方式，但点击<A HREF>标签中的超链接时不会产生表单提交事件，因此隐藏表单域也不支持通用会话跟踪。

## 重写URL

您可以在每个URL后面添加一些额外的数据来区分会话，服务器能够根据这些数据来关联session标识符。

举例来说，http://w3cschool.cc/file.htm;sessionid=12345， session标识符为sessionid=12345，服务器可以用这个数据来识别客户端。

相比而言，重写URL是更好的方式来，就算浏览器不支持cookies也能工作，但缺点是您必须为每个URL动态指定session ID，就算这是个简单的HTML页面。

## session对象

除了以上几种方法外，JSP利用servlet提供的HttpSession接口来识别一个用户，存储这个用户的所有访问信息。

默认情况下，JSP允许会话跟踪，一个新的HttpSession对象将会自动地为新的客户端实例化。禁止会话跟踪需要显式地关掉它，通过将page指令中session属性值设为false来实现，就像下面这样：

```
<%@ page session="false" %>
```

JSP引擎将隐含的session对象暴露给开发者。由于提供了session对象，开发者就可以方便地存储或检索数据。

下表列出了session对象的一些重要方法：

| 方法 | 描述 |
| --- | --- |
| **public Object getAttribute(String name)** | 返回session对象中与指定名称绑定的对象，如果不存在则返回null |
| **public Enumeration getAttributeNames()** | 返回session对象中所有的对象名称 |
| **public long getCreationTime()** | 返回session对象被创建的时间， 以毫秒为单位，从1970年1月1号凌晨开始算起 |
| **public String getId()** | 返回session对象的ID |
| **public long getLastAccessedTime()** | 返回客户端最后访问的时间，以毫秒为单位，从1970年1月1号凌晨开始算起 |
| **public int getMaxInactiveInterval()** | 返回最大时间间隔，以秒为单位，servlet 容器将会在这段时间内保持会话打开 |
| **public void invalidate()** | 将session无效化，解绑任何与该session绑定的对象 |
| **public boolean isNew(** | 返回是否为一个新的客户端，或者客户端是否拒绝加入session |
| **public void removeAttribute(String name)** | 移除session中指定名称的对象 |
| **public void setAttribute(String name, Object value)** | 使用指定的名称和值来产生一个对象并绑定到session中 |
| **public void setMaxInactiveInterval(int interval)** | 用来指定时间，以秒为单位，servlet容器将会在这段时间内保持会话有效 |

# JSP Session应用

这个例子描述了如何使用HttpSession对象来获取创建时间和最后一次访问时间。我们将会为request对象关联一个新的session对象，如果这个对象尚未存在的话。

```
<%@ page import="java.io.*,java.util.*" %>
<%
   // 获取session创建时间
   Date createTime = new Date(session.getCreationTime());
   // 获取最后访问页面的时间
   Date lastAccessTime = new Date(session.getLastAccessedTime());

   String title = "Welcome Back to my website";
   Integer visitCount = new Integer(0);
   String visitCountKey = new String("visitCount");
   String userIDKey = new String("userID");
   String userID = new String("ABCD");

   // 检测网页是否由新的访问用户
   if (session.isNew()){
      title = "Welcome to my website";
      session.setAttribute(userIDKey, userID);
      session.setAttribute(visitCountKey,  visitCount);
   }
   visitCount = (Integer)session.getAttribute(visitCountKey);
   visitCount = visitCount + 1;
   userID = (String)session.getAttribute(userIDKey);
   session.setAttribute(visitCountKey,  visitCount);
%>
<html>
<head>
<title>Session Tracking</title>
</head>
<body>
<center>
<h1>Session Tracking</h1>
</center>
<table border="1" align="center">
<tr bgcolor="#949494">
   <th>Session info</th>
   <th>Value</th>
</tr>
<tr>
   <td>id</td>
   <td><% out.print( session.getId()); %></td>
</tr>
<tr>
   <td>Creation Time</td>
   <td><% out.print(createTime); %></td>
</tr>
<tr>
```

```
    <td>Time of Last Access</td>
    <td><% out.print(lastAccessTime); %></td>
</tr>
<tr>
    <td>User ID</td>
    <td><% out.print(userID); %></td>
</tr>
<tr>
    <td>Number of visits</td>
    <td><% out.print(visitCount); %></td>
</tr>
</table>
</body>
</html>
```

试着访问http://localhost:8080/main.jsp，第一次运行时将会得到如下结果：

Welcome to my website

Session Infomation

| Session info | value |
|---|---|
| id | 0AE3EC93FF44E3C525B4351B77ABB2D5 |
| Creation Time | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| Time of Last Access | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| User ID | ABCD |
| Number of visits | 0 |

再次访问，将会得到如下结果：

Welcome Back to my website

Session Infomation

| info type | value |
|---|---|
| id | 0AE3EC93FF44E3C525B4351B77ABB2D5 |
| Creation Time | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| Time of Last Access | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| User ID | ABCD |
| Number of visits | 1 |

## 删除Session数据

当处理完一个用户的会话数据后，您可以有如下选择：

- 移除一个特定的属性：

  调用public void removeAttribute(String name) 方法来移除指定的属性。

- 删除整个会话：

  调用public void invalidate() 方法来使整个session无效。

- 设置会话有效期：

  调用 public void setMaxInactiveInterval(int interval) 方法来设置session超时。

- 登出用户：

  支持servlet2.4版本的服务器，可以调用 logout()方法来登出用户，并且使所有相关的session无效。

- 配置**web.xml**文件：

  如果使用的是Tomcat，可以向下面这样配置web.xml文件：

```
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

超时以分钟为单位，Tomcat中的默认的超时时间是30分钟。

Servlet中的getMaxInactiveInterval( ) 方法以秒为单位返回超时时间。如果在web.xml中配置的是15分钟，则getMaxInactiveInterval( ) 方法将会返回900。

# JSP 文件上传

JSP可以通过HTML的form表单上传文件到服务器。 文件类型可以是文本文件、二进制文件、 图像文件等其他任何文档。

## 创建文件上传表单

接下来我们使用HTML标签来创建文件上传表单，以下为要注意的点：

- form表单 **method** 属性必须设置为 **POST** 方法， 不能使用 GET 方法。
- form表单 **enctype** 属性需要设置为 **multipart/form-data**。
- form表单 **action** 属性需要设置为提交到后台处理文件上传的jsp文件地址。例如 **uploadFile.jsp** 程序文件用来处理上传的文件。
- 上传文件元素需要使用 <input .../> 标签，属性设置为 type="file"。如果需要上传多个文件， 可以在 <input .../>标签中设置不同的名称。

以下是一个上传文件的表单，实例如下：

```
<html>
<head>
<title>File Uploading Form</title>
</head>
<body>
<h3>File Upload:</h3>
Select a file to upload: <br />
<form action="UploadServlet" method="post"
                        enctype="multipart/form-data">
<input type="file" name="file" size="50" />
<br />
<input type="submit" value="Upload File" />
</form>
</body>
</html>
```

在你本地浏览器访问该文件，显示界面如下所示，在你点击"Upload File"会弹出一个窗口让你选择要上传的文件：

**File Upload:**
Select a file to upload:

选择文件 未选择文件

Upload File

# 后台JSP处理脚本

首先我们先定义文件上传后存储在服务上的位置，你可以将路径写在你的程序当中，或者我们可以在web.xml配置文件中通过设置 context-param 元素来设置文件存储的目录，如下所示：

```
<web-app>
....
<context-param>
    <description>Location to store uploaded file</description>
    <param-name>file-upload</param-name>
    <param-value>
         c:\apache-tomcat-5.5.29\webapps\data\
     </param-value>
</context-param>
....
</web-app>
```

以下脚本文件UploadFile.jsp可以处理多个上传的文件，在使用该脚本前，我们需要注意以下几点：

- 以下实例依赖 FileUpload, 所以你需要在你的classpath中引入最新的 **commons-fileupload.x.x.jar** 包文件。 下载地址为：http://commons.apache.org/fileupload/。
- FileUpload 依赖 Commons IO, 所以你需要在你的classpath中引入最新的 **commons-io-x.x.jar** 。 下载地址为： http://commons.apache.org/io/。
- 在测试以下实例时，你需要上传确认上传的文件大小小于 *maxFileSize* 变量设置的大小 ,否则文件无法上传成功。
- 确保你已经创建了目录 c:\temp 和 c:\apache-tomcat-5.5.29\webapps\data 。

```
<%@ page import="java.io.*,java.util.*, javax.servlet.*" %>
<%@ page import="javax.servlet.http.*" %>
<%@ page import="org.apache.commons.fileupload.*" %>
<%@ page import="org.apache.commons.fileupload.disk.*" %>
<%@ page import="org.apache.commons.fileupload.servlet.*" %>
<%@ page import="org.apache.commons.io.output.*" %>

<%
   File file ;
   int maxFileSize = 5000 * 1024;
   int maxMemSize = 5000 * 1024;
   ServletContext context = pageContext.getServletContext();
   String filePath = context.getInitParameter("file-upload");

   // 验证上传内容了类型
   String contentType = request.getContentType();
   if ((contentType.indexOf("multipart/form-data") >= 0)) {

      DiskFileItemFactory factory = new DiskFileItemFactory();
```

```
        // 设置内存中存储文件的最大值
        factory.setSizeThreshold(maxMemSize);
        // 本地存储的数据大于 maxMemSize.
        factory.setRepository(new File("c:\\temp"));

        // 创建一个新的文件上传处理程序
        ServletFileUpload upload = new ServletFileUpload(factory);
        // 设置最大上传的文件大小
        upload.setSizeMax( maxFileSize );
        try{
           // 解析获取的文件
           List fileItems = upload.parseRequest(request);

           // 处理上传的文件
           Iterator i = fileItems.iterator();

           out.println("<html>");
           out.println("<head>");
           out.println("<title>JSP File upload</title>");
           out.println("</head>");
           out.println("<body>");
           while ( i.hasNext () )
           {
              FileItem fi = (FileItem)i.next();
              if ( !fi.isFormField () )
              {
              // 获取上传文件的参数
              String fieldName = fi.getFieldName();
              String fileName = fi.getName();
              boolean isInMemory = fi.isInMemory();
              long sizeInBytes = fi.getSize();
              // 写入文件
              if( fileName.lastIndexOf("\\") >= 0 ){
              file = new File( filePath ,
              fileName.substring( fileName.lastIndexOf("\\"))) ;
              }else{
              file = new File( filePath ,
              fileName.substring(fileName.lastIndexOf("\\")+1)) ;
              }
              fi.write( file ) ;
              out.println("Uploaded Filename: " + filePath +
              fileName + "<br>");
              }
           }
           out.println("</body>");
           out.println("</html>");
        }catch(Exception ex) {
           System.out.println(ex);
        }
     }else{
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet upload</title>");
```

```
      out.println("</head>");
      out.println("<body>");
      out.println("<p>No file uploaded</p>");
      out.println("</body>");
      out.println("</html>");
   }
%>
```

接下来让我们通过浏览器访问 *http://localhost:8080/UploadFile.htm*，界面如下所示，并上传文件：

**File Upload:**
Select a file to upload:

选择文件 未选择文件

Upload File

如果你的JSP脚本运行正常，文件将被上传至 c:\apache-tomcat-5.5.29\webapps\data\ ，你可以打开文件夹看看是否上传成功。

# JSP 日期处理

使用JSP最重要的优势之一，就是可以使用所有Java API。本章将会详细地讲述Java中的Date类，它在java.util包下，封装了当前日期和时间。

Date类有两个构造函数。第一个构造函数使用当前日期和时间来初始化对象。

```
Date( )
```

第二个构造函数接受一个参数，这个参数表示从1970年1月1日凌晨至所要表示时间的毫秒数。

```
Date(long millisec)
```

获取Date对象后，您就能够使用下表列出的所有方法：

| 方法 | 描述 |
| --- | --- |
| **boolean after(Date date)** | 如果比给定的日期晚，则返回true，否则返回false |
| **boolean before(Date date)** | 如果比给定的日期早，则返回true，否则返回false |
| **Object clone( )** | 获取当前对象的一个副本 |
| **int compareTo(Date date)** | 如果与给定日期相等，则返回0，如果比给定日期早，则返回一个负数，如果比给定日期晚，则返回一个正数 |
| **int compareTo(Object obj)** | 与 compareTo(Date) 方法相同，如果 obj 不是Date类或其子类的对象，抛出ClassCastException异常 |
| **boolean equals(Object date)** | 如果与给定日期相同，则返回true，否则返回false |
| **long getTime( )** | 返回从1970年1月1日凌晨至此对象所表示时间的毫秒数 |
| **int hashCode( )** | 返回此对象的哈希码 |
| **void setTime(long time)** | 使用给定参数设置时间和日期，参数time表示从1970年1月1日凌晨至time所经过的毫秒数 |
| **String toString( )** | 将此对象转换为字符串并返回这个字符串 |

# 获取当前日期和时间

使用JSP编程可以很容易的获取当前日期和时间，只要使用Date对象的toString()方法就行了，就像下面这样：

```
<%@ page import="java.io.*,java.util.*, javax.servlet.*" %>
<html>
<head>
<title>Display Current Date & Time</title>
</head>
<body>
<center>
<h1>Display Current Date & Time</h1>
</center>
<%
   Date date = new Date();
   out.print( "<h2 align=\"center\">" +date.toString()+"</h2>");
%>
</body>
</html>
```

将上面的代码保存在CurrentDate.jsp文件中，然后访问http://localhost:8080/CurrentDate.jsp，运行结果如下：

```
Display Current Date & Time
Mon Jun 21 21:46:49 GMT+04:00 2013
```

刷新http://localhost:8080/CurrentDate.jsp，就可以发现每次刷新所得到的秒数都不相同。

# 日期比较

就像我在开头所提到的，您可以在JSP脚本中使用任何Java方法。如果您想要比较两个日期，

可以参照下面的方法来做：

- 使用getTime()方法得到毫秒数，然后比较毫秒数就行了。
- 使用before()，after()，equals()方法。比如，new Date(99,2,12).before(new Date(99,2,18))返回true。
- 使用compareTo()方法，这个方法在Comparable接口中定义，在Date中实现。

# 使用**SimpleDateFormat**格式化日期

SimpleDateFormat使用一种地区敏感的方式来格式化和解析日期，它允许您使用自定义的模式来格式化日期和时间。

对CurrentDate.jsp稍作修改，得到如下修改后的代码：

```
<%@ page import="java.io.*,java.util.*" %>
<%@ page import="javax.servlet.*,java.text.*" %>
<html>
<head>
<title>Display Current Date & Time</title>
</head>
<body>
<center>
<h1>Display Current Date & Time</h1>
</center>
<%
   Date dNow = new Date( );
   SimpleDateFormat ft =
   new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
   out.print( "<h2 align=\"center\">" + ft.format(dNow) + "</h2>");
%>
</body>
</html>
```

再次编译CurrentDate.jsp，然后访问http://localhost:8080/CurrentDate.jsp，就可以得到如下结果：

```
Display Current Date & Time
Mon 2013.06.21 at 10:06:44 PM GMT+04:00
```

# SimpleDateFormat格式码

要指定模式字符串，需要使用下表列出的格式码：

| 字符 | 描述 | 示例 |
| --- | --- | --- |
| G | 时代标识符 | AD |
| y | 4位数年份 | 2001 |
| M | 月 | July or 07 |
| d | 日 | 10 |
| h | 12小时制， A.M./P.M. (1~12) | 12 |
| H | 24小时制 | 22 |
| m | 分钟 | 30 |
| s | 秒 | 55 |
| S | 毫秒 | 234 |
| E | 星期 | Tuesday |
| D | 一年中的某天 | 360 |
| F | 一个月中某星期的某天 | 2 (second Wed. in July) |
| w | 一年中的某星期 | 40 |
| W | 一个月中的某星期 | 1 |
| a | A.M./P.M. 标记 | PM |
| k | 一天中的某个小时 (1~24) | 24 |
| K | 一天中的某个小时，A.M./P.M. (0~11) | 10 |
| z | 时区 | Eastern Standard Time |
| ' | 文本分隔 | Delimiter |
| " | 单引号 | ` |

更多关于Date类的详细信息请查阅Java API文档。

# JSP 页面重定向

当需要将文档移动到一个新的位置时，就需要使用JSP重定向了。

最简单的重定向方式就是使用response对象的sendRedirect()方法。这个方法的签名如下：

```
public void response.sendRedirect(String location)
throws IOException
```

这个方法将状态码和新的页面位置作为响应发回给浏览器。您也可以使用setStatus()和setHeader()方法来得到同样的效果：

```
....
String site = "http://www.w3cschool.cc" ;
response.setStatus(response.SC_MOVED_TEMPORARILY);
response.setHeader("Location", site);
....
```

## 实例演示

这个例子表明了JSP如何进行页面重定向：

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>Page Redirection</title>
</head>
<body>
<center>
<h1>Page Redirection</h1>
</center>
<%
   // 重定向到新地址
   String site = new String("http://www.w3cschool.cc");
   response.setStatus(response.SC_MOVED_TEMPORARILY);
   response.setHeader("Location", site);
%>
</body>
</html>
```

将以上代码保存在PageRedirecting.jsp文件中，然后访问
http://localhost:8080/PageRedirect.jsp ，它将会把您带至
http://www.w3cschool.cc/。

# JSP 点击量统计

有时候我们需要知道某个页面被访问的次数，这时我们就需要在页面上添加页面统计器，页面访问的统计一般在用户第一次载入时累加该页面的访问数上。

要实现一个计数器，您可以利用应用程序隐式对象和相关方法getAttribute()和setAttribute()来实现。

这个对象表示JSP页面的整个生命周期中。当JSP页面初始化时创建此对象，当JSP页面调用jspDestroy()时删除该对象。

以下是在应用中创建变量的语法：

```
application.setAttribute(String Key, Object Value);
```

您可以使用上述方法来设置一个计数器变量及更新该变量的值。读取该变量的方法如下：

```
application.getAttribute(String Key);
```

在页面每次被访问时，你可以读取计数器的当前值，并递增1，然后重新设置，在下一个用户访问时就将新的值显示在页面上。

## 实例演示

该实例将介绍如何使用JSP来计算特定页面访问的总人数。如果你要计算你网站使用页面的总点击量，那么你就必须将该代码放在所有的JSP页面上。

```
<%@ page import="java.io.*,java.util.*" %>

<html>
<head>
<title>Applcation object in JSP</title>
</head>
<body>
<%
    Integer hitsCount =
      (Integer)application.getAttribute("hitCounter");
    if( hitsCount ==null || hitsCount == 0 ){
       /* 第一次访问 */
       out.println("Welcome to my website!");
       hitsCount = 1;
    }else{
       /* 返回访问值 */
       out.println("Welcome back to my website!");
       hitsCount += 1;
    }
    application.setAttribute("hitCounter", hitsCount);
%>
<center>
<p>Total number of visits: <%= hitsCount%></p>
</center>
</body>
</html>
```

现在我们将上面的代码放置于main.jsp文件上，并访问 *http://localhost:8080/main.jsp*文件。你会看到页面会生成个计数器，在我们每次刷新页面时，计数器都会发生变化（每次刷新增加1）。 你也可以通过不同的浏览器访问，计数器会在每次访问后增加1。如下所示：

```
Welcome back to my website!

Total number of visits: 12
```

# 复位计数器

使用以上方法，在web服务器重启后，计数器会被复位为0，即前面保留的数据都会消失，你可以使用一下几种方式解决该问题:

- 在数据库中定义一个用于统计网页访问量的数据表count，字段为hitcount，hitcount默认值为0，将统计数据写入到数据表中。
- 在每次访问时我们读取表中hitcount字段。
- 每次访问时让hitcount自增1。
- 在页面上显示新的 hitcount 值作为页面的访问量。
- 如果你需要统计每个页面的访问量，你可以使用以上逻辑将代码添加到所有页

面上。

# JSP 自动刷新

想象一下，如果要直播比赛的比分，或股票市场的实时状态，或当前的外汇配给，该怎么实现呢？显然，要实现这种实时功能，您就不得不规律性地刷新页面。

JSP提供了一种机制来使这种工作变得简单，它能够定时地自动刷新页面。

刷新一个页面最简单的方式就是使用response对象的setIntHeader()方法。这个方法的签名如下：

```
public void setIntHeader(String header, int headerValue)
```

这个方法通知浏览器在给定的时间后刷新，时间以秒为单位。

## 页面自动刷新程序示例

这个例子使用了setIntHeader()方法来设置刷新头，模拟一个数字时钟：

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>Auto Refresh Header Example</title>
</head>
<body>
<center>
<h2>Auto Refresh Header Example</h2>
<%
   // Set refresh, autoload time as 5 seconds
   response.setIntHeader("Refresh", 5);
   // Get current time
   Calendar calendar = new GregorianCalendar();
   String am_pm;
   int hour = calendar.get(Calendar.HOUR);
   int minute = calendar.get(Calendar.MINUTE);
   int second = calendar.get(Calendar.SECOND);
   if(calendar.get(Calendar.AM_PM) == 0)
      am_pm = "AM";
   else
      am_pm = "PM";
   String CT = hour+":"+ minute +":"+ second +" "+ am_pm;
   out.println("Crrent Time: " + CT + "\n");
%>
</center>
</body>
</html>
```

把以上代码保存在main.jsp文件中，访问它。它会每隔5秒钟刷新一次页面并获取系统当前时间。运行结果如下：

```
Auto Refresh Header Example
Current Time is: 9:44:50 PM
```

您也可以自己动手写个更复杂点的程序。

# JSP 发送邮件

虽然使用JSP实现邮件发送功能很简单，但是需要有JavaMail API，并且需要安装JavaBean Activation Framework。

- 在这里下载最新版本的 JavaMail。
- 在这里下载最新版本的 JavaBeans Activation Framework(JAF)。

下载并解压这些文件，在根目录下，您将会看到一系列jar包。将mail.jar包和activation.jar包加入CLASSPATH变量中。

## 发送一封简单的邮件

这个例子展示了如何从您的机器发送一封简单的邮件。它假定localhost已经连接至网络并且有能力发送一封邮件。与此同时，请再一次确认mail.jar包和activation.jar包已经添加进CLASSPATH变量中。

```
<%@ page import="java.io.*,java.util.*,javax.mail.*"%>
<%@ page import="javax.mail.internet.*,javax.activation.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%
   String result;
   // 收件人的电子邮件
   String to = "abcd@gmail.com";

   // 发件人的电子邮件
   String from = "mcmohd@gmail.com";

   // 假设你是从本地主机发送电子邮件
   String host = "localhost";

   // 获取系统属性对象
   Properties properties = System.getProperties();

   // 设置邮件服务器
   properties.setProperty("mail.smtp.host", host);

   // 获取默认的Session对象。
   Session mailSession = Session.getDefaultInstance(properties);

   try{
      // 创建一个默认的MimeMessage对象。
      MimeMessage message = new MimeMessage(mailSession);
      // 设置 From: 头部的header字段
      message.setFrom(new InternetAddress(from));
      // 设置 To: 头部的header字段
      message.addRecipient(Message.RecipientType.TO,
```

```
                                new InternetAddress(to));
        // 设置 Subject: header字段
        message.setSubject("This is the Subject Line!");
        // 现在设置的实际消息
        message.setText("This is actual message");
        // 发送消息
        Transport.send(message);
        result = "Sent message successfully....";
     }catch (MessagingException mex) {
        mex.printStackTrace();
        result = "Error: unable to send message....";
     }
%>
<html>
<head>
<title>Send Email using JSP</title>
</head>
<body>
<center>
<h1>Send Email using JSP</h1>
</center>
<p align="center">
<%
   out.println("Result: " + result + "\n");
%>
</p>
</body>
</html>
```

现在访问 http://localhost:8080/SendEmail.jsp ，它将会发送一封邮件给 abcd@gmail.com 并显示如下结果：

```
Send Email using JSP
Result: Sent message successfully....
```

如果想要把邮件发送给多人，下面列出的方法可以用来指明多个邮箱地址：

```
void addRecipients(Message.RecipientType type,
                   Address[] addresses)
throws MessagingException
```

参数的描述如下：

- type：这个值将会被设置成TO，CC,或BCC。CC代表副本，BCC代表黑色副本，例子程序中使用的是TO。
- addresses：这是一个邮箱地址的数组，当指定邮箱地址时需要使用InternetAddress()方法。

# 发送一封HTML邮件

这个例子发送一封简单的HTML邮件。它假定您的localhost已经连接至网络并且有能力发送邮件。与此同时，请再一次确认mail.jar包和activation.jar包已经添加进CLASSPATH变量中。

这个例子和前一个例子非常相似，不过在这个例子中我们使用了setContent()方法，将"text/html"做为第二个参数传给它，用来表明消息中包含了HTML内容。

```
<%@ page import="java.io.*,java.util.*,javax.mail.*"%>
<%@ page import="javax.mail.internet.*,javax.activation.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%
   String result;
   // 收件人的电子邮件
   String to = "abcd@gmail.com";

   // 发件人的电子邮件
   String from = "mcmohd@gmail.com";

   // 假设你是从本地主机发送电子邮件
   String host = "localhost";

   // 获取系统属性对象
   Properties properties = System.getProperties();

   // 设置邮件服务器
   properties.setProperty("mail.smtp.host", host);

   // 获取默认的Session对象。
   Session mailSession = Session.getDefaultInstance(properties);

   try{
      // 创建一个默认的MimeMessage对象。
      MimeMessage message = new MimeMessage(mailSession);
      // 设置 From: 头部的header字段
      message.setFrom(new InternetAddress(from));
      // 设置 To: 头部的header字段
      message.addRecipient(Message.RecipientType.TO,
                               new InternetAddress(to));
      // 设置 Subject: header字段
      message.setSubject("This is the Subject Line!");

      // 设置 HTML消息
      message.setContent("<h1>This is actual message</h1>",
                            "text/html" );
      // 发送消息
      Transport.send(message);
      result = "Sent message successfully....";
   }catch (MessagingException mex) {
      mex.printStackTrace();
```

```
        result = "Error: unable to send message....";
    }
%>
<html>
<head>
<title>Send HTML Email using JSP</title>
</head>
<body>
<center>
<h1>Send Email using JSP</h1>
</center>
<p align="center">
<%
    out.println("Result: " + result + "\n");
%>
</p>
</body>
</html>
```

现在你可以尝试使用以上JSP文件来发送HTML消息的电子邮件。

## 在邮件中包含附件

这个例子告诉我们如何发送一封包含附件的邮件。

```
<%@ page import="java.io.*,java.util.*,javax.mail.*"%>
<%@ page import="javax.mail.internet.*,javax.activation.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%
    String result;
    // 收件人的电子邮件
    String to = "abcd@gmail.com";

    // 发件人的电子邮件
    String from = "mcmohd@gmail.com";

    // 假设你是从本地主机发送电子邮件
    String host = "localhost";

    // 获取系统属性对象
    Properties properties = System.getProperties();

    // 设置邮件服务器
    properties.setProperty("mail.smtp.host", host);

    // 获取默认的Session对象。
    Session mailSession = Session.getDefaultInstance(properties);

    try{
        // 创建一个默认的MimeMessage对象。
```

```
        MimeMessage message = new MimeMessage(mailSession);

        // 设置 From: 头部的header字段
        message.setFrom(new InternetAddress(from));

        // 设置 To: 头部的header字段
        message.addRecipient(Message.RecipientType.TO,
                                 new InternetAddress(to));

        // 设置 Subject: header字段
        message.setSubject("This is the Subject Line!");

        // 创建消息部分
        BodyPart messageBodyPart = new MimeBodyPart();

        // 填充消息
        messageBodyPart.setText("This is message body");

        // 创建多媒体消息
        Multipart multipart = new MimeMultipart();

        // 设置文本消息部分
        multipart.addBodyPart(messageBodyPart);

        // 附件部分
        messageBodyPart = new MimeBodyPart();
        String filename = "file.txt";
        DataSource source = new FileDataSource(filename);
        messageBodyPart.setDataHandler(new DataHandler(source));
        messageBodyPart.setFileName(filename);
        multipart.addBodyPart(messageBodyPart);

        // 发送完整消息
        message.setContent(multipart );

        // 发送消息
        Transport.send(message);
        String title = "Send Email";
        result = "Sent message successfully....";
     }catch (MessagingException mex) {
        mex.printStackTrace();
        result = "Error: unable to send message....";
     }
%>
<html>
<head>
<title>Send Attachement Email using JSP</title>
</head>
<body>
<center>
<h1>Send Attachement Email using JSP</h1>
</center>
<p align="center">
```

```
<%
   out.println("Result: " + result + "\n");
%>
</p>
</body>
</html>
```

# 用户认证部分

如果邮件服务器需要用户名和密码来进行用户认证的话，可以像下面这样来设置：

```
 props.setProperty("mail.user", "myuser");
 props.setProperty("mail.password", "mypwd");
```

# 使用表单发送邮件

使用HTML表单接收一封邮件，并通过request对象获取所有邮件信息：

```
String to = request.getParameter("to");
String from = request.getParameter("from");
String subject = request.getParameter("subject");
String messageText = request.getParameter("body");
```

获取以上信息后，您就可以使用前面提到的例子来发送邮件了。

# JSP 高级教程

# JSP 标准标签库（JSTL）

JSP标准标签库（JSTL）是一个JSP标签集合，它封装了JSP应用的通用核心功能。

JSTL支持通用的、结构化的任务，比如迭代，条件判断，XML文档操作，国际化标签，SQL标签。 除了这些，它还提供了一个框架来使用集成JSTL的自定义标签。

根据JSTL标签所提供的功能，可以将其分为5个类别。

- 核心标签
- 格式化标签
- **SQL** 标签
- **XML** 标签
- **JSTL** 函数

## JSTL 库安装

Apache Tomcat安装JSTL 库步骤如下：

- 从Apache的标准标签库中下载的二进包(jakarta-taglibs-standard-current.zip)。下载地址：http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/
- 下载jakarta-taglibs-standard-1.1.1.zip 包并解压，将jakarta-taglibs-standard-1.1.1/lib/下的两个jar文件：standard.jar和jstl.jar文件拷贝到/WEB-INF/lib/下。

使用任何库，你必须在每个JSP文件中的头部包含<taglib>标签。

## 核心标签

核心标签是最常用的JSTL标签。引用核心标签库的语法如下：

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core" %>
```

| 标签 | 描述 |
| --- | --- |
| <c:out> | 用于在JSP中显示数据，就像<%= ... > |
| <c:set> | 用于保存数据 |
| <c:remove> | 用于删除数据 |
| <c:catch> | 用来处理产生错误的异常状况，并且将错误信息储存起来 |
| <c:if> | 与我们在一般程序中用的if一样 |
| <c:choose> | 本身只当做<c:when>和<c:otherwise>的父标签 |
| <c:when> | <c:choose>的子标签，用来判断条件是否成立 |
| <c:otherwise> | <c:choose>的子标签，接在<c:when>标签后，当<c:when>标签判断为false时被执行 |
| <c:import> | 检索一个绝对或相对 URL，然后将其内容暴露给页面 |
| <c:forEach> | 基础迭代标签，接受多种集合类型 |
| <c:forTokens> | 根据指定的分隔符来分隔内容并迭代输出 |
| <c:param> | 用来给包含或重定向的页面传递参数 |
| <c:redirect> | 重定向至一个新的URL. |
| <c:url> | 使用可选的查询参数来创造一个URL |

## 格式化标签

JSTL格式化标签用来格式化并输出文本、日期、时间、数字。引用格式化标签库的语法如下：

```
<%@ taglib prefix="fmt"
           uri="http://java.sun.com/jsp/jstl/fmt" %>
```

| 标签 | 描述 |
| --- | --- |
| <fmt:formatNumber> | 使用指定的格式或精度格式化数字 |
| <fmt:parseNumber> | 解析一个代表着数字，货币或百分比的字符串 |
| <fmt:formatDate> | 使用指定的风格或模式格式化日期和时间 |
| <fmt:parseDate> | 解析一个代表着日期或时间的字符串 |
| <fmt:bundle> | 绑定资源 |
| <fmt:setLocale> | 指定地区 |
| <fmt:setBundle> | 绑定资源 |
| <fmt:timeZone> | 指定时区 |
| <fmt:setTimeZone> | 指定时区 |
| <fmt:message> | 显示资源配置文件信息 |
| <fmt:requestEncoding> | 设置request的字符编码 |

# SQL标签

JSTL SQL标签库提供了与关系型数据库（Oracle，MySQL，SQL Server等等）进行交互的标签。引用SQL标签库的语法如下：

```
<%@ taglib prefix="sql"
           uri="http://java.sun.com/jsp/jstl/sql" %>
```

| 标签 | 描述 |
| --- | --- |
| <sql:setDataSource> | 指定数据源 |
| <sql:query> | 运行SQL查询语句 |
| <sql:update> | 运行SQL更新语句 |
| <sql:param> | 将SQL语句中的参数设为指定值 |
| <sql:dateParam> | 将SQL语句中的日期参数设为指定的java.util.Date 对象值 |
| <sql:transaction> | 在共享数据库连接中提供嵌套的数据库行为元素，将所有语句以一个事务的形式来运行 |

# XML 标签

JSTL XML标签库提供了创建和操作XML文档的标签。引用XML标签库的语法如下：

```
<%@ taglib prefix="x"
           uri="http://java.sun.com/jsp/jstl/xml" %>
```

在使用xml标签前，你必须将XML 和 XPath 的相关包拷贝至你的<Tomcat 安装目录>\lib下:

XercesImpl.jar:

下载地址： http://www.apache.org/dist/xerces/j/

xalan.jar:

下载地址： http://xml.apache.org/xalan-j/index.html

| 标签 | 描述 |
|---|---|
| <x:out> | 与<%= ... >,类似，不过只用于XPath表达式 |
| <x:parse> | 解析 XML 数据 |
| <x:set> | 设置XPath表达式 |
| <x:if> | 判断XPath表达式，若为真，则执行本体中的内容，否则跳过本体 |
| <x:forEach> | 迭代XML文档中的节点 |
| <x:choose> | <x:when>和<x:otherwise>的父标签 |
| <x:when> | <x:choose>的子标签，用来进行条件判断 |
| <x:otherwise> | <x:choose>的子标签，当<x:when>判断为false时被执行 |
| <x:transform> | 将XSL转换应用在XML文档中 |
| <x:param> | 与<x:transform>共同使用，用于设置XSL样式表 |

# JSTL函数

JSTL包含一系列标准函数，大部分是通用的字符串处理函数。引用JSTL函数库的语法如下：

```
<%@ taglib prefix="fn"
           uri="http://java.sun.com/jsp/jstl/functions" %>
```

| 函数 | 描述 |
|---|---|
| fn:contains() | 测试输入的字符串是否包含指定的子串 |
| fn:containsIgnoreCase() | 测试输入的字符串是否包含指定的子串，大小写不敏感 |
| fn:endsWith() | 测试输入的字符串是否以指定的后缀结尾 |
| fn:escapeXml() | 跳过可以作为XML标记的字符 |
| fn:indexOf() | 返回指定字符串在输入字符串中出现的位置 |
| fn:join() | 将数组中的元素合成一个字符串然后输出 |
| fn:length() | 返回字符串长度 |
| fn:replace() | 将输入字符串中指定的位置替换为指定的字符串然后返回 |
| fn:split() | 将字符串用指定的分隔符分隔然后组成一个子字符串数组并返回 |
| fn:startsWith() | 测试输入字符串是否以指定的前缀开始 |
| fn:substring() | 返回字符串的子集 |
| fn:substringAfter() | 返回字符串在指定子串之后的子集 |
| fn:substringBefore() | 返回字符串在指定子串之前的子集 |
| fn:toLowerCase() | 将字符串中的字符转为小写 |
| fn:toUpperCase() | 将字符串中的字符转为大写 |
| fn:trim() | 移除首位的空白符 |

# JSP 连接数据库

本章节假设您已经对JDBC有一定的了解。在开始学习JSP数据库访问前，请确保JDBC环境已经正确配置。

首先，让我们按照下面的步骤来创建一个简单的表并插入几条简单的记录：

## 创建表

在数据库中创建一个Employees表，步骤如下：

### 步骤1：

打开CMD，然后进入数据库安装目录：

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

### 步骤2：

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ********
mysql>
```

### 步骤3：

在TEST数据库中创建Employee表：

```
mysql> use TEST;
mysql> create table Employees
    (
     id int not null,
     age int not null,
     first varchar (255),
     last varchar (255)
    );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

# 插入数据记录

创建好Employee表后，往表中插入几条记录：

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)

mysql>
```

# SELECT操作

接下来的这个例子告诉我们如何使用JSTL SQL标签来运行SQL SELECT语句：

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>SELECT Operation</title>
</head>
<body>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/TEST"
     user="root"  password="pass123"/>

<sql:query dataSource="${snapshot}" var="result">
SELECT * from Employees;
</sql:query>

<table border="1" width="100%">
<tr>
   <th>Emp ID</th>
   <th>First Name</th>
   <th>Last Name</th>
   <th>Age</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
   <td><c:out value="${row.id}"/></td>
   <td><c:out value="${row.first}"/></td>
   <td><c:out value="${row.last}"/></td>
   <td><c:out value="${row.age}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

访问这个JSP例子，运行结果如下：

| Emp ID | First Name | Last Name | Age |
| --- | --- | --- | --- |
| 100 | Zara | Ali | 18 |
| 101 | Mahnaz | Fatma | 25 |
| 102 | Zaid | Khan | 30 |
| 103 | Sumit | Mittal | 28 |

# INSERT操作

这个例子告诉我们如何使用JSTL SQL标签来运行SQL INSERT语句：

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>JINSERT Operation</title>
</head>
<body>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/TEST"
     user="root"  password="pass123"/>

<sql:update dataSource="${snapshot}" var="result">
INSERT INTO Employees VALUES (104, 2, 'Nuha', 'Ali');
</sql:update>

<sql:query dataSource="${snapshot}" var="result">
SELECT * from Employees;
</sql:query>

<table border="1" width="100%">
<tr>
   <th>Emp ID</th>
   <th>First Name</th>
   <th>Last Name</th>
   <th>Age</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
   <td><c:out value="${row.id}"/></td>
   <td><c:out value="${row.first}"/></td>
   <td><c:out value="${row.last}"/></td>
   <td><c:out value="${row.age}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

访问这个JSP例子，运行结果如下：

| Emp ID | First Name | Last Name | Age |
|---|---|---|---|
| 100 | Zara | Ali | 18 |
| 101 | Mahnaz | Fatma | 25 |
| 102 | Zaid | Khan | 30 |
| 103 | Sumit | Mittal | 28 |
| 104 | Nuha | Ali | 2 |

# DELETE操作

这个例子告诉我们如何使用JSTL SQL标签来运行SQL DELETE语句：

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>DELETE Operation</title>
</head>
<body>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/TEST"
     user="root"  password="pass123"/>

<c:set var="empId" value="103"/>

<sql:update dataSource="${snapshot}" var="count">
  DELETE FROM Employees WHERE Id = ?
  <sql:param value="${empId}" />
</sql:update>

<sql:query dataSource="${snapshot}" var="result">
   SELECT * from Employees;
</sql:query>

<table border="1" width="100%">
<tr>
   <th>Emp ID</th>
   <th>First Name</th>
   <th>Last Name</th>
   <th>Age</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
   <td><c:out value="${row.id}"/></td>
   <td><c:out value="${row.first}"/></td>
   <td><c:out value="${row.last}"/></td>
   <td><c:out value="${row.age}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

访问这个JSP例子，运行结果如下：

| Emp ID | First Name | Last Name | Age |
| --- | --- | --- | --- |
| 100 | Zara | Ali | 18 |
| 101 | Mahnaz | Fatma | 25 |
| 102 | Zaid | Khan | 30 |

# UPDATE操作

这个例子告诉我们如何使用JSTL SQL标签来运行SQL UPDATE语句：

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>DELETE Operation</title>
</head>
<body>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/TEST"
     user="root"  password="pass123"/>

<c:set var="empId" value="102"/>

<sql:update dataSource="${snapshot}" var="count">
  UPDATE Employees SET last = 'Ali'
  <sql:param value="${empId}" />
</sql:update>

<sql:query dataSource="${snapshot}" var="result">
   SELECT * from Employees;
</sql:query>

<table border="1" width="100%">
<tr>
   <th>Emp ID</th>
   <th>First Name</th>
   <th>Last Name</th>
   <th>Age</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
   <td><c:out value="${row.id}"/></td>
   <td><c:out value="${row.first}"/></td>
   <td><c:out value="${row.last}"/></td>
   <td><c:out value="${row.age}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

访问这个JSP例子，运行结果如下：

| Emp ID | First Name | Last Name | Age |
| --- | --- | --- | --- |
| 100 | Zara | Ali | 18 |
| 101 | Mahnaz | Fatma | 25 |
| 102 | Zaid | Ali | 30 |

# JSP XML 数据处理

当通过HTTP发送XML数据时，就有必要使用JSP来处理传入和流出的XML文档了，比如RSS文档。作为一个XML文档，它仅仅只是一堆文本而已，使用JSP创建XML文档并不比创建一个HTML文档难。

## 使用JSP发送XML

使用JSP发送XML内容就和发送HTML内容一样。唯一的不同就是您需要把页面的context属性设置为text/xml。要设置context属性，使用<%@page % >命令，就像这样：

```
<%@ page contentType="text/xml" %>
```

接下来这个例子向浏览器发送XML内容：

```
<%@ page contentType="text/xml" %>

<books>
   <book>
      <name>Padam History</name>
      <author>ZARA</author>
      <price>100</price>
   </book>
</books>
```

使用不同的浏览器来访问这个例子，看看这个例子所呈现的文档树。

## 在JSP中处理XML

在使用JSP处理XML之前，您需要将与XML 和XPath相关的两个库文件放在<Tomcat Installation Directory>\lib目录下：

- XercesImpl.jar：在这下载http://www.apache.org/dist/xerces/j/
- xalan.jar：在这下载http://xml.apache.org/xalan-j/index.html

books.xml文件:

```
<books>
<book>
  <name>Padam History</name>
  <author>ZARA</author>
  <price>100</price>
</book>
<book>
  <name>Great Mistry</name>
  <author>NUHA</author>
  <price>2000</price>
</book>
</books>
```

main.jsp文件：

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

<html>
<head>
  <title>JSTL x:parse Tags</title>
</head>
<body>
<h3>Books Info:</h3>
<c:import var="bookInfo" url="http://localhost:8080/books.xml"/>

<x:parse xml="${bookInfo}" var="output"/>
<b>The title of the first book is</b>:
<x:out select="$output/books/book[1]/name" />
<br>
<b>The price of the second book</b>:
<x:out select="$output/books/book[2]/price" />

</body>
</html>
```

访问http://localhost:8080/main.jsp，运行结果如下：

```
BOOKS INFO:
The title of the first book is:Padam History
The price of the second book: 2000
```

# 使用JSP格式化XML

这个是XSLT样式表style.xsl文件：

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="html" indent="yes"/>

<xsl:template match="/">
  <html>
  <body>
   <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="books">
  <table border="1" width="100%">
    <xsl:for-each select="book">
      <tr>
        <td>
          <i><xsl:value-of select="name"/></i>
        </td>
        <td>
          <xsl:value-of select="author"/>
        </td>
        <td>
          <xsl:value-of select="price"/>
        </td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>
</xsl:stylesheet>
```

这个是main.jsp文件：

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

<html>
<head>
  <title>JSTL x:transform Tags</title>
</head>
<body>
<h3>Books Info:</h3>
<c:set var="xmltext">
  <books>
    <book>
      <name>Padam History</name>
      <author>ZARA</author>
      <price>100</price>
    </book>
    <book>
      <name>Great Mistry</name>
      <author>NUHA</author>
      <price>2000</price>
    </book>
  </books>
</c:set>

<c:import url="http://localhost:8080/style.xsl" var="xslt"/>
<x:transform xml="${xmltext}" xslt="${xslt}"/>

</body>
</html>
```

运行结果如下：

BOOKS INFO:

| | | |
|---|---|---|
| *Padam History* | ZARA | 100 |
| *Great Mistry* | NUHA | 2000 |

更多关于使用JSTL处理XML的内容请查阅JSP标准标签库。

# JSP JavaBean

JavaBean是特殊的Java类，使用J ava语言书写，并且遵守JavaBeans API规范。

接下来给出的是JavaBean与其它Java类相比而言独一无二的特征：

- 提供一个默认的无参构造函数。
- 需要被序列化并且实现了Serializable接口。
- 可能有一系列可读写属性。
- 可能有一系列的"getter"或"setter"方法。

## JavaBeans属性

一个JavaBean对象的属性应该是可访问的。这个属性可以是任意合法的Java数据类型，包括自定义Java类。

一个JavaBean对象的属性可以是可读写，或只读，或只写。JavaBean对象的属性通过JavaBean实现类中提供的两个方法来访问：

| 方法 | 描述 |
|---|---|
| get**PropertyName**() | 举例来说，如果属性的名称为myName，那么这个方法的名字就要写成getMyName()来读取这个属性。这个方法也称为访问器。 |
| set**PropertyName**() | 举例来说，如果属性的名称为myName，那么这个方法的名字就要写成setMyName()来写入这个属性。这个方法也称为写入器。 |

一个只读的属性只提供getPropertyName()方法，一个只写的属性只提供setPropertyName()方法。

## JavaBeans程序示例

这是StudentBean.java文件：

```
package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable
{
   private String firstName = null;
   private String lastName = null;
   private int age = 0;

   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

编译StudentBean.java文件，在本章最后的例子中将会使用到它。

# 访问JavaBeans

<jsp:useBean>标签可以在JSP中声明一个JavaBean，然后使用。声明后，JavaBean对象就成了脚本变量，可以通过脚本元素或其他自定义标签来访问。<jsp:useBean>标签的语法格式如下：

```
<jsp:useBean id="bean's name" scope="bean's scope" typeSpec/>
```

其中，根据具体情况，scope的值可以是page，request，session或application。id值可任意只要不和同一JSP文件中其它<jsp:useBean>中id值一样就行了。

接下来给出的是<jsp:useBean>标签的一个简单的用法：

```
<html>
<head>
<title>useBean Example</title>
</head>
<body>

<jsp:useBean id="date" class="java.util.Date" />
<p>The date/time is <%= date %>

</body>
</html>
```

它将会产生如下结果：

```
The date/time is Thu Sep 30 11:18:11 GST 2013
```

## 访问JavaBeans对象的属性

在<jsp:useBean>标签主体中使用<jsp:getProperty/>标签来调用getter方法，使用<jsp:setProperty/>标签来调用setter方法，语法格式如下：

```
<jsp:useBean id="id" class="bean's class" scope="bean's scope">
   <jsp:setProperty name="bean's id" property="property name"
                    value="value"/>
   <jsp:getProperty name="bean's id" property="property name"/>
   ...........
</jsp:useBean>
```

name属性指的是Bean的id属性。property属性指的是想要调用的getter或setter方法。

接下来给出使用以上语法进行属性访问的一个简单例子：

```
<html>
<head>
<title>get and set properties Example</title>
</head>
<body>

<jsp:useBean id="students"
                    class="com.tutorialspoint.StudentsBean">
   <jsp:setProperty name="students" property="firstName"
                    value="Zara"/>
   <jsp:setProperty name="students" property="lastName"
                    value="Ali"/>
   <jsp:setProperty name="students" property="age"
                    value="10"/>
</jsp:useBean>

<p>Student First Name:
   <jsp:getProperty name="students" property="firstName"/>
</p>
<p>Student Last Name:
   <jsp:getProperty name="students" property="lastName"/>
</p>
<p>Student Age:
   <jsp:getProperty name="students" property="age"/>
</p>

</body>
</html>
```

将StudentBean.class加入CLASSPATH环境变量中，然后访问以上JSP，运行结果如下：

```
Student First Name: Zara

Student Last Name: Ali

Student Age: 10
```

# JSP 自定义标签

自定义标签是用户定义的JSP语言元素。当JSP页面包含一个自定义标签时将被转化为servlet，标签转化为对被 称为tag handler的对象的操作，即当servlet执行时Web container调用那些操作。

JSP标签扩展可以让你创建新的标签并且可以直接插入到一个JSP页面。 JSP 2.0规范中引入Simple Tag Handlers来编写这些自定义标记。

你可以继承SimpleTagSupport类并重写的doTag()方法来开发一个最简单的自定义标签。

## 创建"Hello"标签

接下来，我们想创建一个自定义标签叫作<ex:Hello>，标签格式为：

```
<ex:Hello />
```

要创建自定义的JSP标签，你首先必须创建处理标签的Java类。所以，让我们创建一个HelloTag类，如下所示：

```
package com.tutorialspoint;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HelloTag extends SimpleTagSupport {

  public void doTag() throws JspException, IOException {
    JspWriter out = getJspContext().getOut();
    out.println("Hello Custom Tag!");
  }
}
```

以下代码重写了doTag()方法，方法中使用了getJspContext()方法来获取当前的JspContext对象，并将"Hello Custom Tag!"传递给JspWriter对象。

编译以上类，并将其复制到环境变量CLASSPATH目录中。最后创建如下标签库：<Tomcat安装目录>webapps\ROOT\WEB-INF\custom.tld。

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>com.tutorialspoint.HelloTag</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

接下来，我们就可以在JSP文件中使用Hello标签：

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <ex:Hello/>
  </body>
</html>
```

以上程序输出结果为：

```
Hello Custom Tag!
```

# 访问标签体

你可以像标准标签库一样在标签中包含消息内容。如我们要在我们自定义的Hello中包含内容，格式如下：

```
<ex:Hello>
   This is message body
</ex:Hello>
```

我们可以修改标签处理类文件，代码如下：

```
package com.tutorialspoint;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HelloTag extends SimpleTagSupport {

   StringWriter sw = new StringWriter();
   public void doTag()
      throws JspException, IOException
    {
       getJspBody().invoke(sw);
       getJspContext().getOut().println(sw.toString());
    }

}
```

接下来我们需要修改TLD文件，如下所示：

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD with Body</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>com.tutorialspoint.HelloTag</tag-class>
    <body-content>scriptless</body-content>
  </tag>
</taglib>
```

现在我们可以在JSP使用修改后的标签，如下所示:

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <ex:Hello>
        This is message body
    </ex:Hello>
  </body>
</html>
```

以上程序输出结果如下所示：

```
This is message body
```

# 自定义标签属性

你可以在自定义标准中设置各种属性，要接收属性，值自定义标签类必须实现setter方法， JavaBean 中的setter方法如下所示：

```
package com.tutorialspoint;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HelloTag extends SimpleTagSupport {

   private String message;

   public void setMessage(String msg) {
      this.message = msg;
   }

   StringWriter sw = new StringWriter();

   public void doTag()
      throws JspException, IOException
    {
       if (message != null) {
          /* 从属性中使用消息 */
          JspWriter out = getJspContext().getOut();
          out.println( message );
       }
       else {
          /* 从内容体中使用消息 */
          getJspBody().invoke(sw);
          getJspContext().getOut().println(sw.toString());
       }
   }

}
```

属性的名称是"message"，所以setter方法??是的setMessage()。现在让我们在TLD文件中使用的<attribute>元素添加此属性：

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD with Body</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>com.tutorialspoint.HelloTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
       <name>message</name>
    </attribute>
  </tag>
</taglib>
```

现在我们就可以在JSP文件中使用message属性了，如下所示：

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <ex:Hello message="This is custom tag" />
  </body>
</html>
```

以上实例数据输出结果为：

```
This is custom tag
```

你还可以包含以下属性：

| 属性 | 描述 |
|---|---|
| name | 定义属性的名称。每个标签的是属性名称必须是唯一的。 |
| required | 指定属性是否是必须的或者可选的,如果设置为false为可选。 |
| rtexprvalue | 声明在运行表达式时，标签属性是否有效。 |
| type | 定义该属性的Java类类型 。默认指定为 **String** |
| description | 描述信息 |
| fragment | 如果声明了该属性,属性值将被视为一个 **JspFragment**。 |

以下是指定相关的属性实例：

```
.....
    <attribute>
      <name>attribute_name</name>
      <required>false</required>
      <type>java.util.Date</type>
      <fragment>false</fragment>
    </attribute>
.....
```

如果你使用了两个属性，修改TLD文件，如下所示：

```
.....
    <attribute>
      <name>attribute_name1</name>
      <required>false</required>
      <type>java.util.Boolean</type>
      <fragment>false</fragment>
    </attribute>
    <attribute>
      <name>attribute_name2</name>
      <required>true</required>
      <type>java.util.Date</type>
    </attribute>
.....
```

# JSP 表达式语言

JSP表达式语言（EL）使得访问存储在JavaBean中的数据变得非常简单。JSP EL既可以用来创建算术表达式也可以用来创建逻辑表达式。在JSP EL表达式内可以使用整型数，浮点数，字符串，常量true、false，还有null。

## 一个简单的语法

典型的，当您需要在JSP标签中指定一个属性值时，只需要简单地使用字符串即可：

```
<jsp:setProperty name="box" property="perimeter" value="100"/>
```

JSP EL允许您指定一个表达式来表示属性值。一个简单的表达式语法如下：

```
${expr}
```

其中，expr指的是表达式。在JSP EL中通用的操作符是"."和"[]"。这两个操作符允许您通过内嵌的JSP对象访问各种各样的JavaBean属性。

举例来说，上面的<jsp:setProperty>标签可以使用表达式语言改写成如下形式：

```
<jsp:setProperty name="box" property="perimeter"
                 value="${2*box.width+2*box.height}"/>
```

当JSP编译器在属性中见到"${}"格式后，它会产生代码来计算这个表达式，并且产生一个替代品来代替表达式的值。

您也可以在标签的模板文本中使用表达式语言。比如<jsp:text>标签简单地将其主体中的文本插入到JSP输出中：

```
<jsp:text>
<h1>Hello JSP!</h1>
</jsp:text>
```

现在，在<jsp:text>标签主体中使用表达式，就像这样：

```
<jsp:text>
Box Perimeter is: ${2*box.width + 2*box.height}
</jsp:text>
```

在EL表达式中可以使用圆括号来组织子表达式。比如${(1 + 2) 3}等于9，但是${1 + (2 3)} 等于7。

想要停用对EL表达式的评估的话，需要使用page指令将isELIgnored属性值设为true：

```
<%@ page isELIgnored ="true|false" %>
```

这样，EL表达式就会被忽略。若设为false，则容器将会计算EL表达式。

## EL中的基础操作符

EL表达式支持大部分Java所提供的算术和逻辑操作符：

| 操作符 | 描述 |
|---|---|
| . | 访问一个Bean属性或者一个映射条目 |
| [] | 访问一个数组或者链表的元素 |
| ( ) | 组织一个子表达式以改变优先级 |
| + | 加 |
| - | 减或负 |
| * | 乘 |
| / or div | 除 |
| % or mod | 取模 |
| == or eq | 测试是否相等 |
| != or ne | 测试是否不等 |
| < or lt | 测试是否小于 |
| > or gt | 测试是否大于 |
| <= or le | 测试是否小于等于 |
| >= or gt | 测试是否大于等于 |
| && or and | 测试逻辑与 |
| \|\| or or | 测试逻辑或 |
| ! or not | 测试取反 |
| empty | 测试是否空值 |

# JSP EL中的函数

JSP EL允许您在表达式中使用函数。这些函数必须被定义在自定义标签库中。函数的使用语法如下：

```
${ns:func(param1, param2, ...)}
```

ns指的是命名空间（namespace），func指的是函数的名称，param1指的是第一个参数，param2指的是第二个参数，以此类推。比如，有函数fn:length，在JSTL库中定义，可以像下面这样来获取一个字符串的长度：

```
${fn:length("Get my length")}
```

要使用任何标签库中的函数，您需要将这些库安装在服务器中，然后使用<taglib>标签在JSP文件中包含这些库。

# JSP EL隐含对象

JSP EL支持下表列出的隐含对象：

| 隐含对象 | 描述 |
|---|---|
| pageScope | page 作用域 |
| requestScope | request 作用域 |
| sessionScope | session 作用域 |
| applicationScope | application 作用域 |
| param | Request 对象的参数，字符串 |
| paramValues | Request对象的参数，字符串集合 |
| header | HTTP 信息头，字符串 |
| headerValues | HTTP 信息头，字符串集合 |
| initParam | 上下文初始化参数 |
| cookie | Cookie值 |
| pageContext | 当前页面的pageContext |

您可以在表达式中使用这些对象，就像使用变量一样。接下来会给出几个例子来更好的理解这个概念。

## pageContext对象

pageContext对象是JSP中pageContext对象的引用。通过pageContext对象，您可以访问request对象。比如，访问request对象传入的查询字符串，就像这样：

```
${pageContext.request.queryString}
```

## Scope对象

pageScope，requestScope，sessionScope，applicationScope变量用来访问存储在各个作用域层次的变量。

举例来说，如果您需要显式访问在applicationScope层的box变量，可以这样来访问：applicationScope.box。

## param和paramValues对象

param和paramValues对象用来访问参数值，通过使用request.getParameter方法和request.getParameterValues方法。

举例来说，访问一个名为order的参数，可以这样使用表达式：${param.order}，或者${param["order"]}。

接下来的例子表明了如何访问request中的username参数：

```
<%@ page import="java.io.*,java.util.*" %>
<%
    String title = "Accessing Request Param";
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>${param["username"]}</p>
</div>
</body>
</html>
```

param对象返回单一的字符串，而paramValues对象则返回一个字符串数组。

# header和headerValues对象

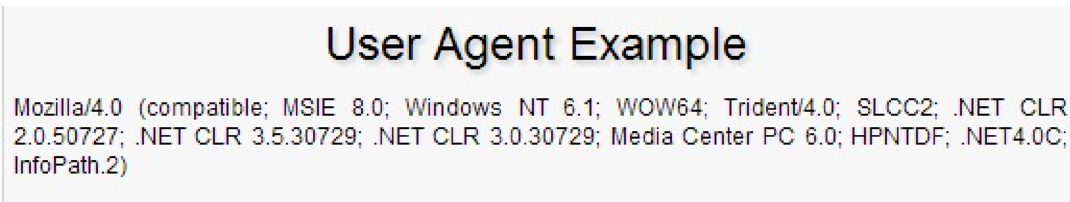
header和headerValues对象用来访问信息头，通过使用 request.getHeader方法和request.getHeaders方法。

举例来说，要访问一个名为user-agent的信息头，可以这样使用表达式：${header.user-agent}，或者${header["user-agent"]}。

接下来的例子表明了如何访问user-agent信息头：

```
<%@ page import="java.io.*,java.util.*" %>
<%
    String title = "User Agent Example";
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>${header["user-agent"]}</p>
</div>
</body>
</html>
```

运行结果如下：




header对象返回单一值，而headerValues则返回一个字符串数组。

# JSP 异常处理

当编写JSP程序的时候，程序员可能会遗漏一些BUG，这些BUG可能会出现在程序的任何地方。JSP代码中通常有以下几类异常:

- 检查型异常:检查型异常就是一个典型的用户错误或者一个程序员无法预见的错误。举例来说，如果一个文件将要被打开，但是无法找到这个文件，则一个异常被抛出。这些异常不能再编译期被简单地忽略。
- 运行时异常:一个运行时异常可能已经被程序员避免，这种异常在编译期将会被忽略。
- 错误:这里没有异常，但问题是它超出了用户或者程序员的控制范围。错误通常会在代码中被忽略，您几乎不能拿它怎么样。举例来或，栈溢出错误。这些错误都会在编译期被忽略。

本节将会给出几个简单而优雅的方式来处理运行时异常和错误。

## 使用Exception对象

exception对象是Throwable子类的一个实例，只在错误页面中可用。下表列出了Throwable类中一些重要的方法:

| 方法 | 描述 |
|---|---|
| **public String getMessage()** | 返回异常的信息。这个信息在Throwable构造函数中被初始化 |
| **public ThrowablegetCause()** | 返回引起异常的原因，类型为Throwable对象 |
| **public String toString()** | 返回类名 |
| **public void printStackTrace()** | 将异常栈轨迹输出至System.err |
| **public StackTraceElement [] getStackTrace()** | 以栈轨迹元素数组的形式返回异常栈轨迹 |
| **public ThrowablefillInStackTrace()** | 使用当前栈轨迹填充Throwable对象 |

JSP提供了可选项来为每个JSP页面指定错误页面。无论何时页面抛出了异常，JSP容器都会自动地调用错误页面。

接下来的例子为main.jsp指定了一个错误页面。使用<%@page errorPage="XXXXX"%>指令指定一个错误页面。

```
<%@ page errorPage="ShowError.jsp" %>

<html>
<head>
   <title>Error Handling Example</title>
</head>
<body>
<%
   // Throw an exception to invoke the error page
   int x = 1;
   if (x == 1)
   {
      throw new RuntimeException("Error condition!!!");
   }
%>
</body>
</html>
```

现在，编写ShowError.jsp文件如下:

```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>Show Error Page</title>
</head>
<body>
<h1>Opps...</h1>
<p>Sorry, an error occurred.</p>
<p>Here is the exception stack trace: </p>
<pre>
<% exception.printStackTrace(response.getWriter()); %>
```

注意到，ShowError.jsp文件使用了<%@page isErrorPage="true"%>指令，这个指令告诉JSP编译器需要产生一个异常实例变量。

现在试着访问main.jsp页面，它将会产生如下结果:

```
java.lang.RuntimeException: Error condition!!!
......

Opps...
Sorry, an error occurred.

Here is the exception stack trace:
```

# 在错误页面中使用JSTL标签

可以利用JSTL标签来编写错误页面ShowError.jsp。这个例子中的代码与上例代码的逻辑几乎一样，但是本例的代码有更好的结构，并且能够提供更多信息:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page isErrorPage="true" %>
<html>
<head>
<title>Show Error Page</title>
</head>
<body>
<h1>Opps...</h1>
<table width="100%" border="1">
<tr valign="top">
<td width="40%"><b>Error:</b></td>
<td>${pageContext.exception}</td>
</tr>
<tr valign="top">
<td><b>URI:</b></td>
<td>${pageContext.errorData.requestURI}</td>
</tr>
<tr valign="top">
<td><b>Status code:</b></td>
<td>${pageContext.errorData.statusCode}</td>
</tr>
<tr valign="top">
<td><b>Stack trace:</b></td>
<td>
<c:forEach var="trace"
         items="${pageContext.exception.stackTrace}">
<p>${trace}</p>
</c:forEach>
</td>
</tr>
</table>
</body>
</html>
```

运行结果如下:

Opps...

| | |
|---|---|
| Error: | java.lang.RuntimeException: Error condition!!! |
| URI: | /main.jsp |
| Status code: | 500 |
| Stack trace: | org.apache.jsp.main_jsp._jspService(main_jsp.java:65)<br>org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:68)<br>javax.servlet.http.HttpServlet.service(HttpServlet.java:722)<br>org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)<br>javax.servlet.http.HttpServlet.service(HttpServlet.java:722)<br>................... |

# 使用 try…catch块

如果您想要将异常处理放在一个页面中，并且对不同的异常进行不同的处理，那么您就需要使用try…catch块了。

接下来的这个例子显示了如何使用try…catch块，将这些代码放在main.jsp中:

```
<html>
<head>
   <title>Try...Catch Example</title>
</head>
<body>
<%
   try{
      int i = 1;
      i = i / 0;
      out.println("The answer is " + i);
   }
   catch (Exception e){
      out.println("An exception occurred: " + e.getMessage());
   }
%>
</body>
</html>
```

试着访问main.jsp，它将会产生如下结果:

```
An exception occurred: / by zero
```

# JSP 调试

要测试/调试一个JSP或servlet程序总是那么的难。JSP和Servlets程序趋向于牵涉到大量客户端/服务器之间的交互，这很有可能会产生错误，并且很难重现出错的环境。

接下来将会给出一些小技巧和小建议，来帮助您调试程序。

## 使用System.out.println()

System.out.println()可以很方便地标记一段代码是否被执行。当然，我们也可以打印出各种各样的值。此外：

- 自从System对象成为Java核心对象后，它便可以使用在任何地方而不用引入额外的类。使用范围包括Servlets，JSP，RMI，EJB's，Beans，类和独立应用。
- 与在断点处停止运行相比，用System.out进行输出不会对应用程序的运行流程造成重大的影响，这个特点在定时机制非常重要的应用程序中就显得非常有用了。

接下来给出了使用System.out.println()的语法：

```
System.out.println("Debugging message");
```

这是一个使用System.out.print()的简单例子：

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>System.out.println</title></head>
<body>
<c:forEach var="counter" begin="1" end="10" step="1" >
   <c:out value="${counter-5}"/></br>
   <% System.out.println( "counter= " +
                     pageContext.findAttribute("counter") ); %>
</c:forEach>
</body>
</html>
```

现在，如果运行上面的例子的话，它将会产生如下的结果：

```
-4
-3
-2
-1
0
1
2
3
4
5
```

如果使用的是Tomcat服务器，您就能够在logs目录下的stdout.log文件中发现多出了如下内容：

```
counter=1
counter=2
counter=3
counter=4
counter=5
counter=6
counter=7
counter=8
counter=9
counter=10
```

使用这种方法可以将变量和其它的信息输出至系统日志中，用来分析并找出造成问题的深层次原因。

# 使用JDB Logger

J2SE日志框架可为任何运行在JVM中的类提供日志记录服务。因此我们可以利用这个框架来记录任何信息。

让我们来重写以上代码，使用JDK中的 logger API：

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page import="java.util.logging.Logger" %>

<html>
<head><title>Logger.info</title></head>
<body>
<% Logger logger=Logger.getLogger(this.getClass().getName());%>

<c:forEach var="counter" begin="1" end="10" step="1" >
   <c:set var="myCount" value="${counter-5}" />
   <c:out value="${myCount}"/></br>
   <% String message = "counter="
                  + pageContext.findAttribute("counter")
                  + " myCount="
                  + pageContext.findAttribute("myCount");
                  logger.info( message );
   %>
</c:forEach>
</body>
</html>
```

它的运行结果与先前的类似，但是，它可以获得额外的信息输出至stdout.log文件中。在这我们使用了logger中的info方法。下面我们给出stdout.log文件中的一个快照：

```
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=1 myCount=-4
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=2 myCount=-3
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=3 myCount=-2
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=4 myCount=-1
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=5 myCount=0
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=6 myCount=1
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=7 myCount=2
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=8 myCount=3
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=9 myCount=4
24-Sep-2013 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=10 myCount=5
```

消息可以使用各种优先级发送，通过使用sever()，warning()，info()，config()，fine()，finer()，finest()方法。finest()方法用来记录最好的信息，而sever()方法用来记录最严重的信息。

使用Log4J 框架来将消息记录在不同的文件中，这些消息基于严重程度和重要性来进行分类。

## 调试工具

NetBeans是树形结构，是开源的Java综合开发环境，支持开发独立的Java应用程序和网络应用程序，同时也支持JSP调试。

NetBeans支持如下几个基本的调试功能：

- 断点
- 单步跟踪
- 观察点

详细的信息可以查看NetBeans使用手册。

# 使用JDB Debugger

可以在JSP和servlets中使用jdb命令来进行调试，就像调试普通的应用程序一样。

通常，我们直接调试sun.servlet.http.HttpServer 对象来查看HttpServer在响应HTTP请求时执行JSP/Servlets的情况。这与调试applets非常相似。不同之处在于，applets程序实际调试的是sun.applet.AppletViewer。

大部分调试器在调试applets时都能够自动忽略掉一些细节，因为它知道如何调试applets。如果想要将调试对象转移到JSP身上，就需要做好以下两点：

- 设置调试器的classpath，让它能够找到sun.servlet.http.Http-Server 和相关的类。
- 设置调试器的classpath，让它能够找到您的JSP文件和相关的类。

设置好classpath后，开始调试sun.servlet.http.Http-Server 。您可以在JSP文件的任意地方设置断点，只要你喜欢，然后使用浏览器发送一个请求给服务器就应该可以看见程序停在了断点处。

## 使用注释

程序中的注释在很多方面都对程序的调试起到一定的帮助作用。注释可以用在调试程序的很多方面中。

JSP使用Java注释。如果一个BUG消失了，就请仔细查看您刚注释过的代码，通常都能找出原因。

## 客户端和服务器的头模块

有时候，当JSP没有按照预定的方式运行时，查看未加工的HTTP请求和响应也是很有用的。如果对HTTP的结构很熟悉的话，您可以直接观察request和response然后看看这些头模块到底怎么了。

## 重要调试技巧

这里我们再透露两个调试JSP的小技巧：

- 使用浏览器显示原始的页面内容，用来区分是否是格式问题。这个选项通常在View菜单下。
- 确保浏览器在强制重新载入页面时没有捕获先前的request输出。若使用的是Netscape Navigator浏览器，则用Shift-Reload；若使用的是IE浏览器，则用Shift-Refresh。

# JSP 国际化

在开始前，需要解释几个重要的概念：

- 国际化（i18n）：表明一个页面根据访问者的语言或国家来呈现不同的翻译版本。
- 本地化（l10n）：向网站添加资源，以使它适应不同的地区和文化。比如网站的印度语版本。
- 区域：这是一个特定的区域或文化，通常认为是一个语言标志和国家标志通过下划线连接起来。比如"en_US"代表美国英语地区。

如果想要建立一个全球化的网站，就需要关心一系列项目。本章将会详细告诉您如何处理国际化问题，并给出了一些例子来加深理解。

JSP容器能够根据request的locale属性来提供正确地页面版本。接下来给出了如何通过request对象来获得Locale对象的语法：

```
java.util.Locale request.getLocale()
```

## 检测Locale

下表列举出了Locale对象中比较重要的方法，用于检测request对象的地区，语言，和区域。所有这些方法都会在浏览器中显示国家名称和语言名称：

| 方法 | 描述 |
|---|---|
| **String getCountry()** | 返回国家/地区码的英文大写，或 ISO 3166 2-letter 格式的区域 |
| **String getDisplayCountry()** | 返回要显示给用户的国家名称 |
| **String getLanguage()** | 返回语言码的英文小写，或ISO 639 格式的区域 |
| **String getDisplayLanguage()** | 返回要给用户看的语言名称 |
| **String getISO3Country()** | 返回国家名称的3字母缩写 |
| **String getISO3Language()** | 返回语言名称的3字母缩写 |

## 实例演示

这个例子告诉我们如何在JSP中显示语言和国家：

```
<%@ page import="java.io.*,java.util.Locale" %>
<%@ page import="javax.servlet.*,javax.servlet.http.* "%>
<%
   //获取客户端本地化信息
   Locale locale = request.getLocale();
   String language = locale.getLanguage();
   String country = locale.getCountry();
%>
<html>
<head>
<title>Detecting Locale</title>
</head>
<body>
<center>
<h1>Detecting Locale</h1>
</center>
<p align="center">
<%
   out.println("Language : " + language  + "<br />");
   out.println("Country  : " + country   + "<br />");
%>
</p>
</body>
</html>
```

## 语言设置

JSP可以使用西欧语言来输出一个页面，比如英语，西班牙语，德语，法语，意大利语等等。由此可见，设置Content-Language信息头来正确显示所有字符是很重要的。

第二点就是，需要使用HTML字符实体来显示特殊字符，比如"ñ" 代表的是"?"，"¡"代表的是 "?" ：

```
<%@ page import="java.io.*,java.util.Locale" %>
<%@ page import="javax.servlet.*,javax.servlet.http.* "%>
<%
    // Set response content type
    response.setContentType("text/html");
    // Set spanish language code.
    response.setHeader("Content-Language", "es");
    String title = "En Espa?ol";

%>
<html>
<head>
<title><%  out.print(title); %></title>
</head>
<body>
<center>
<h1><%  out.print(title); %></h1>
</center>
<div align="center">
<p>En Espa?ol</p>
<p>?Hola Mundo!</p>
</div>
</body>
</html>
```

## 区域特定日期

可以使用java.text.DateFormat类和它的静态方法getDateTimeInstance()来格式化日期和时间。接下来的这个例子显示了如何根据指定的区域来格式化日期和时间：

```
<%@ page import="java.io.*,java.util.Locale" %>
<%@ page import="javax.servlet.*,javax.servlet.http.* "%>
<%@ page import="java.text.DateFormat,java.util.Date" %>

<%
    String title = "Locale Specific Dates";
    //Get the client's Locale
    Locale locale = request.getLocale( );
    String date = DateFormat.getDateTimeInstance(
                                  DateFormat.FULL,
                                  DateFormat.SHORT,
                                  locale).format(new Date( ));
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>Local Date: <%  out.print(date); %></p>
</div>
</body>
</html>
```

## 区域特定货币

可以使用java.text.NumberFormat类和它的静态方法getCurrencyInstance()来格式化数字。比如在区域特定货币中的long型和double型。接下来的例子显示了如何根据指定的区域来格式化货币：

```
<%@ page import="java.io.*,java.util.Locale" %>
<%@ page import="javax.servlet.*,javax.servlet.http.* "%>
<%@ page import="java.text.NumberFormat,java.util.Date" %>

<%
    String title = "Locale Specific Currency";
    //Get the client's Locale
    Locale locale = request.getLocale( );
    NumberFormat nft = NumberFormat.getCurrencyInstance(locale);
    String formattedCurr = nft.format(1000000);
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>Formatted Currency: <%  out.print(formattedCurr); %></p>
</div>
</body>
</html>
```

# 区域特定百分比

可以使用java.text.NumberFormat类和它的静态方法getPercentInstance()来格式化百分比。接下来的例子告诉我们如何根据指定的区域来格式化百分比：

```
<%@ page import="java.io.*,java.util.Locale" %>
<%@ page import="javax.servlet.*,javax.servlet.http.* "%>
<%@ page import="java.text.NumberFormat,java.util.Date" %>

<%
    String title = "Locale Specific Percentage";
    //Get the client's Locale
    Locale locale = request.getLocale( );
    NumberFormat nft = NumberFormat.getPercentInstance(locale);
    String formattedPerc = nft.format(0.51);
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>Formatted Percentage: <%  out.print(formattedPerc); %></p>
</div>
</body>
</html>
```

# Log4j教程

log4j是一个用Java编写的可靠，快速和灵活的日志框架（API），它在Apache软件许可下发布。 Log4j已经被移植到了C，C++，C＃，Perl，Python和Ruby等语言中。Log4j是高度可配置的，并可通过在运行时的外部文件配置。它根据记录的优先级别，并提供机制，以指示记录信息到许多的目的地，诸如：数据库，文件，控制台，UNIX系统日志等。

Log4j中有三个主要组成部分：

- **loggers:** 负责捕获记录信息。
- **appenders :** 负责发布日志信息，以不同的首选目的地。
- **layouts:** 负责格式化不同风格的日志信息。

## log4j的历史

- 始于1996年初的E.U. SEMPER （安全电子市场为欧洲）跟踪API的项目。
- 不计其数的改进，在几个大量的工作之后，API已经发展成为log4j，一个流行的日志记录包为Java。
- 该软件包是Apache软件许可证，由开源认证是一个不折不扣的开源许可证下发布。
- 最新log4j的版本，包括完整的源代码，类文件和文档可以在这里找到 http://logging.apache.org/log4j/.

## log4j 特性:

- log4j的是线程安全的
- log4j是经过优化速度的
- log4j是基于一个名为记录器的层次结构
- log4j的支持每个记录器多输出追加器（appender）
- log4j支持国际化。
- log4j并不限于一组预定义的设备
- 日志行为可以使用配置文件在运行时设置
- log4j设计从一开始就是处理Java异常

- log4j使用多个层次，即ALL，TRACE，DEBUG，INFO，WARN，ERROR和FATAL
- 日志输出的格式可以通过扩展Layout类容易地改变
- 日志输出的目标，以及在写入策略可通过实现Appender程序接口改变
- log4j 会故障停止。然而，尽管它肯定努力确保传递，log4j不保证每个日志语句将被传递到目的地。

## 日志记录N个缺点及优点：

日志是软件开发的重要组成部分。一个精心编写的日志代码提供快速的调试，维护方便，以及应用程序的运行时信息结构化存储。

日志记录确实也有它的缺点。它可以减缓的应用程序。如果太详细，它可能会导致滚动失明。为了减轻这些影响，log4j被设计为是可靠，快速和可扩展。

由于记录很少为应用的主要重点，但log4j API致力于成为易于理解和使用。

# log4j安装配置 - Log4j教程

Log4j的API包使用Apache软件许可证，由开源倡议认证一个完全成熟的开源许可证下发布。

最新log4j的版本，包括完整的源代码，类文件和文档可以在这里找到 http://logging.apache.org/log4j/.

下载 apache-log4j-x.x.x.tar.gz 做到以下几点：

## 步骤1:

将下载的文件解压缩和解压在 /usr/local/ 目录，如下所示：

```
$ gunzip apache-log4j-1.2.15.tar.gz
$ tar -xvf apache-log4j-1.2.15.tar
apache-log4j-1.2.15/tests/input/
apache-log4j-1.2.15/tests/input/xml/
apache-log4j-1.2.15/tests/src/
apache-log4j-1.2.15/tests/src/java/
apache-log4j-1.2.15/tests/src/java/org/
.......................................
```

当执行解压缩，这将创建一个名称 apache-log4j-x.x.x 的目录层次结构如下：

```
-rw-r--r--  1 root root   3565 2007-08-25 00:09 BUILD-INFO.txt
-rw-r--r--  1 root root   2607 2007-08-25 00:09 build.properties.sa
-rw-r--r--  1 root root  32619 2007-08-25 00:09 build.xml
drwxr-xr-x 14 root root   4096 2010-02-04 14:09 contribs
drwxr-xr-x  5 root root   4096 2010-02-04 14:09 examples
-rw-r--r--  1 root root   2752 2007-08-25 00:09 INSTALL
-rw-r--r--  1 root root   4787 2007-08-25 00:09 KEYS
-rw-r--r--  1 root root  11366 2007-08-25 00:09 LICENSE
-rw-r--r--  1 root root 391834 2007-08-25 00:29 log4j-1.2.15.jar
-rw-r--r--  1 root root    160 2007-08-25 00:09 NOTICE
-rwxr-xr-x  1 root root  10240 2007-08-25 00:27 NTEventLogAppender.
-rw-r--r--  1 root root  17780 2007-08-25 00:09 pom.xml
drwxr-xr-x  7 root root   4096 2007-08-25 00:13 site
drwxr-xr-x  8 root root   4096 2010-02-04 14:08 src
drwxr-xr-x  6 root root   4096 2010-02-04 14:09 tests
```

## 步骤2:

这一步是可选的，取决于什么功能，要使用log4j 框架。如果已经有以下安装在您的机器，那么可以使用这些软件包，否则将需要安装它们，才能正常使log4j工作

- JavaMail API: 电子邮件。基于log4j日志记录功能需要Java邮件API（mail.jar）在计算机上安装 https://glassfish.dev.java.net/javaee5/mail/
- JavaBeans 活动框架： Java邮件API还需要JavaBeans激活框架（的activation.jar）从http://java.sun.com/products/javabeans/jaf/index.jsp 上下载安装在您的计算机上
- Java 信息服务: log4j的JMS兼容的功能将需要两个JMS和JNDI（Java命名和Directory接口），可以从http://java.sun.com/products/jms 下载安装
- XML Parser: 需要一个JAXP兼容的XML解析器来使用log4j。请确保已经在机器上安装xerces.jar， 可从http://xerces.apache.org/xerces-j/install.html 下载安装

## 步骤3:

这一步是非常重要的，需要设置CLASSPATH和PATH变量正确。在这里要设置只是log4j.x.x.x.jar 文件

```
$ pwd
/usr/local/apache-log4j-1.2.15
$ export CLASSPATH=
      $CLASSPATH:/usr/local/apache-log4j-1.2.15/log4j-1.2.15.jar
$ export PATH=$PATH:/usr/local/apache-log4j-1.2.15/
```

注意：如果在Window上开发使用Eclipse的话，可以在Eclipse创建用户库并加入到构建路径中。

# log4j架构 - Log4j教程

Log4j API设计为分层结构，其中每一层提供了不同的对象，对象执行不同的任务。这使得设计灵活，根据将来需要来扩展。

有两种类型可用在Log4j的框架对象。

- 核心对象： 框架的强制对象和框架的使用。
- 支持对象： 框架和支持体核心对象，可选的对象执行另外重要的任务。

## 核心对象：

### Logger对象：

顶级层的Logger，它提供Logger对象。Logger对象负责捕获日志信息及它们存储在一个空间的层次结构。

### 布局对象：

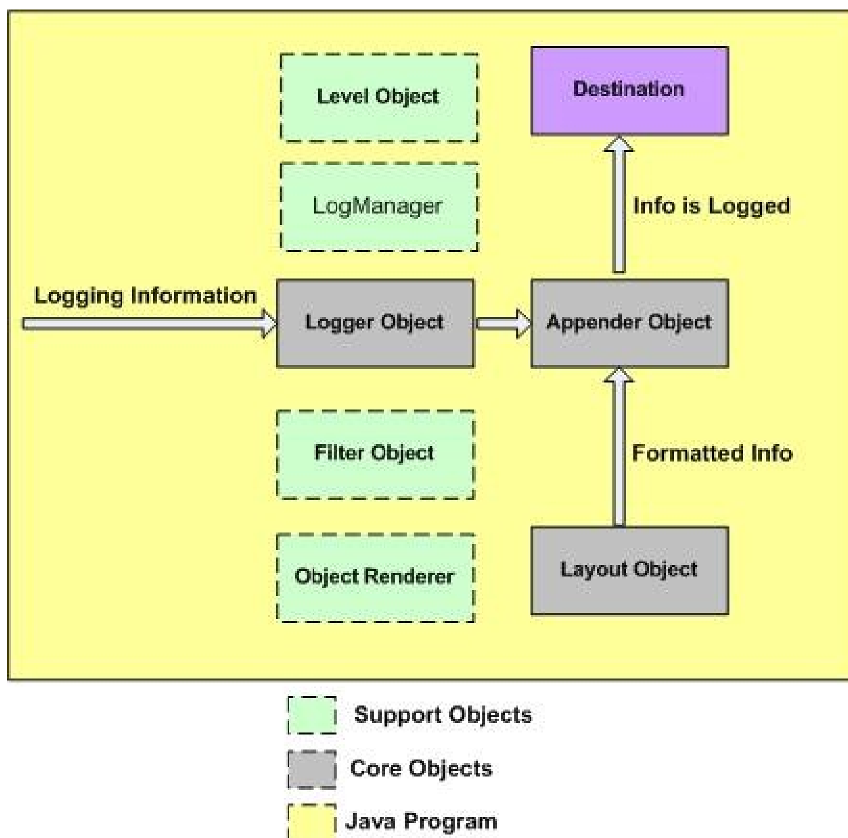
该层提供其用于格式化不同风格的日志信息的对象。布局层提供支持Appender对象到发布日志信息之前。

布局对象的发布方式是人类可读的及可重复使用的记录信息的一个重要的角色。

### Appender对象：

下位层提供Appender对象。Appender对象负责发布日志信息，以不同的首选目的地，如数据库，文件，控制台，UNIX系统日志等。

以下是显示Log4J框架的不同组件的虚拟图：




# 支持对象：

log4j框架的其他重要的对象起到日志框架的一个重要作用：

## Level对象：

级别对象定义的任何记录信息的粒度和优先级。有记录的七个级别在API中定义：OFF, DEBUG, INFO, ERROR, WARN, FATAL 和 ALL

## Filter对象：

过滤对象用于分析日志信息及是否应记录或不用这些信息做出进一步的决定。

一个appender对象可以有与之关联的几个Filter对象。如果日志记录信息传递给特定Appender对象，都和特定Appender相关的Filter对象批准的日志信息，然后才能发布到所连接的目的地。

## 对象渲染器：

ObjectRenderer对象是一个指定提供传递到日志框架的不同对象的字符串表示。这个对象所使用的布局对象来准备最后的日志信息。

## 日志管理：

日志管理对象管理的日志框架。它负责从一个系统级的配置文件或配置类读取初始配置参数。

# log4j配置 - Log4j教程

上一章介绍log4j的核心组件。本章介绍如何使用配置文件来配置这些核心组件。配置log4j涉及分配级别，定义追加程序，并在配置文件中指定布局的对象。

log4j.properties文件是一个键 - 值对保存 log4j 配置属性文件。默认情况下，日志管理在CLASSPATH 查找一个名为 log4j.properties 的文件。

- 根日志记录器的级别定义为DEBUG并连接附加器命名为X到它
- 设置名为X的附加目的地是一个有效的appender
- 设置布局的附加器X

## log4j.properties 语法:

以下是 log4j.properties 文件的一个appender X的语法：

```
# Define the root logger with appender X
log4j.rootLogger = DEBUG, X

# Set the appender named X to be a File appender
log4j.appender.X=org.apache.log4j.FileAppender

# Define the layout for X appender
log4j.appender.X.layout=org.apache.log4j.PatternLayout
log4j.appender.X.layout.conversionPattern=%m%n
```

## log4j.properties 示例:

使用上面的语法，我们定义 log4j.properties 文件如下：

- 根日志记录器(logger)的级别定义为DEBUG并连接附加器命名为FILE
- 附加器(appender)File是定义为org.apache.log4j.FileAppender并写入到一个名为“log.out”位于日志log目录下
- 定义的布局模式是%m%n，这意味着每打印日志消息之后，将加上一个换行符

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

需要注意的是log4j支持UNIX风格的变量替换，如 ${variableName}.

## 调试级别：

使用DEBUG两个追加程序。所有可能的选项有：

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- FATAL
- ALL

这些级别将在 Log4j调试级别 这一文章中解释

# Appenders:

Apache的log4j提供Appender对象主要负责打印日志消息到不同的目的地，如控制台，文件，sockets，NT事件日志等等。

每个Appender对象具有与之相关联的不同的属性，并且这些属性表明对象的行为

| 属性 | 描述 |
|---|---|
| layout | Appender使用布局Layout 对象和与之相关的格式化的日志记录信息转换模式 |
| target | 目标可以是一个控制台，一个文件，或根据附加器的另一个项目 |
| level | 级别是必需的，以控制日志消息的过滤 |
| threshold | Appender可以有与之独立的记录器级别相关联的级别阈值水平。Appender忽略具有级别低于阈级别的任何日志消息 |
| filter | Filter 对象可以分析超出级别的匹配记录信息，并决定是否记录的请求应该由一个特定 Appender 或忽略处理 |

可以通过包括以下方法的配置文件中的下面设置一个 Appender 对象添加到记录器：

```
log4j.logger.[logger-name]=level, appender1,appender..n
```

可以编写以XML格式相同的结构如下：

```
<logger name="com.apress.logging.log4j" additivity="false">
   <appender-ref ref="appender1"/>
   <appender-ref ref="appender2"/>
</logger>
```

如果想要添加Appender对象到程序，那么可以使用下面的方法：

```
public void addAppender(Appender appender);
```

addAppender()方法添加一个appender到Logger对象。作为示例配置演示，可以添加很多Appender对象到记录器在逗号分隔的列表，每个打印日志信息分离目的地。

我们仅使用一个附加目的地FileAppender在我们上面的例子。所有可能的附加目的地选项有：

- AppenderSkeleton
- AsyncAppender
- ConsoleAppender
- DailyRollingFileAppender
- ExternallyRolledFileAppender
- FileAppender

- JDBCAppender
- JMSAppender
- LF5Appender
- NTEventLogAppender
- NullAppender
- RollingFileAppender
- SMTPAppender
- SocketAppender
- SocketHubAppender
- SyslogAppender
- TelnetAppender
- WriterAppender

我们将涵盖 FileAppender文件 和 JDBCAppender 记录将被包括记录在数据库

## Layout:

我们使用的PatternLayout 使用 appender。所有可能的选项有：

- DateLayout
- HTMLLayout
- PatternLayout
- SimpleLayout
- XMLLayout

使用HTMLLayout和XMLLayout，可以在HTML和XML格式和生成日志。

## 布局格式：

如何在章节格式的日志信息：Log格式

# log4j示例程序 - Log4j教程

前面我们已经看到了如何创建一个配置文件。本教程将讲解如何生成调试信息和日志在一个简单的文本文件。

下面是我们的例子中创建了一个简单的配置文件。这里再重复一次：

- 下载最新的Log4j库：http://logging.apache.org/log4j/2.x/download.html
- 根记录器的级别定义为DEBUG并连接appender命名为FILE。
- appender FILE文件被定义为 org.apache.log4j.FileAppender 并写入到一个名为“log.out”位于 log 目录下。
- 定义的布局模式是 %m%n，这意味着打印日志消息之后自动加上一个换行符。

所以 log4j.properties 文件的内容如下：

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

## 在Java程序中使用log4j：

下面的Java类是一个非常简单的例子，Java应用程序初始化，然后使用Log4J日志库。

```
import org.apache.log4j.Logger;

import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class log4jExample{
  /* Get actual class name to be printed on */
  static Logger log = Logger.getLogger(
                      log4jExample.class.getName());

  public static void main(String[] args)
                throws IOException,SQLException{

     log.debug("Hello this is an debug message");
     log.info("Hello this is an info message");
  }
}
```

## 编译和运行：

下面是步骤编译并运行上述程序。确保在进行编译和执行之前，适当地设置PATH和CLASSPATH。

所有的库应该在 CLASSPATH 和 log4j.properties 文件应该在PATH可用。所以，做到以下几点：

- 创建log4j.properties如上图所示。
- 创建log4jExample.java如上图所示，并对其进行编译。
- 执行log4jExample二进制运行程序。

在里面 /usr/home/log4j/log.out 文件会得到下面的结果：

```
Hello this is an debug message
Hello this is an info message
```

# log4j Logger方法 - Log4j教程

Logger类提供了多种方法来处理日志活动。 Logger类不允许实例化一个新的记录器实例，但它提供了两个静态方法获得一个 Logger 对象：

- **public static Logger getRootLogger();**
- **public static Logger getLogger(String name);**

此处两种方法的第一个返回应用程序实例根记录器并没有名字。任何其他命名的Logger对象实例是通过第二种方法通过记录器的名称获得。记录器名称是可以传递任何字符串，通常是类或包的名称，因为我们已经使用在最后一章。

```
static Logger log = Logger.getLogger(log4jExample.class.getName());
```

## Logging 方法:

我们得到了一个名为记录器的实例之后，可以使用记录的几种方法来记录消息。Logger类有专门用于打印日志信息下面的方法如下。

| SN | 方法及描述 |
|---|---|
| 1 | **public void debug(Object message)** 这种方法打印使用 Level.DEBUG 消息级别 |
| 2 | **public void error(Object message)** 这种方法打印使用 Level.ERROR 消息级别 |
| 3 | **public void fatal(Object message);** 这种方法打印使用 Level.FATAL 消息级别 |
| 4 | **public void info(Object message);** 这种方法打印使用 Level.INFO 消息级别 |
| 5 | **public void warn(Object message);** 这种方法打印使用 Level.WARN 消息级别 |
| 6 | **public void trace(Object message);** 这种方法打印使用Level.TRACE消息级别 |

所有的级别定义在org.apache.log4j.Level类中，并且任何上述方法都可以调用如下：

```
import org.apache.log4j.Logger;

public class LogClass {
   private static org.apache.log4j.Logger log = Logger
                                 .getLogger(LogClass.class);
   public static void main(String[] args) {
      log.trace("Trace Message!");
      log.debug("Debug Message!");
      log.info("Info Message!");
      log.warn("Warn Message!");
      log.error("Error Message!");
      log.fatal("Fatal Message!");
   }
}
```

当编译并运行LogClass程序会产生以下结果：

```
Debug Message!
Info Message!
Warn Message!
Error Message!
Fatal Message!
```

所有的调试消息更有意义，当它们在级别组合使用。级别将在下一章介绍，那么在下一节会有一个很好的理解及如何使用这些方法在不同的级别调试。

# log4j日志记录级别 - Log4j教程

org.apache.log4j.Level类提供以下级别，但也可以通过Level类的子类自定义级别。

| Level | 描述 |
| --- | --- |
| ALL | 各级包括自定义级别 |
| DEBUG | 指定细粒度信息事件是最有用的应用程序调试 |
| ERROR | 错误事件可能仍然允许应用程序继续运行 |
| FATAL | 指定非常严重的错误事件，这可能导致应用程序中止 |
| INFO | 指定能够突出在粗粒度级别的应用程序运行情况的信息的消息 |
| OFF | 这是最高等级，为了关闭日志记录 |
| TRACE | 指定细粒度比DEBUG更低的信息事件 |
| WARN | 指定具有潜在危害的情况 |

## 日志级别是如何工作？

级别p的级别使用q，在记录日志请求时，如果p>=q启用。这条规则是log4j的核心。它假设级别是有序的。对于标准级别它们关系如下：ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF。

下面的例子明确指出如何可以过滤所有的DEBUG和INFO消息。这个程序使用记录并执行setLevel（Level.X）方法来设置所需的日志记录级别：

这个例子将打印，除了调试和信息的所有消息：

```
import org.apache.log4j.*;

public class LogClass {
   private static org.apache.log4j.Logger log = Logger
                                 .getLogger(LogClass.class);
   public static void main(String[] args) {
      log.setLevel(Level.WARN);

      log.trace("Trace Message!");
      log.debug("Debug Message!");
      log.info("Info Message!");
      log.warn("Warn Message!");
      log.error("Error Message!");
      log.fatal("Fatal Message!");
   }
}
```

当编译并运行LogClass程序会产生以下结果：

```
Warn Message!
Error Message!
Fatal Message!
```

## 使用配置文件设置级别：

Log4j提供这些可以让程序员自由更改源代码，改变调试级别的配置级别是基于文件设置。

以下是上面的例子使用 log.setLevel（Level.WARN）方法的配置文件与上面的例子例子功能一样。

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = WARN, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

现在，使用下面的程序：

```
import org.apache.log4j.*;

public class LogClass {
   private static org.apache.log4j.Logger log = Logger
                                    .getLogger(LogClass.class);
   public static void main(String[] args) {
      log.trace("Trace Message!");
      log.debug("Debug Message!");
      log.info("Info Message!");
      log.warn("Warn Message!");
      log.error("Error Message!");
      log.fatal("Fatal Message!");
   }
}
```

现在，编译和运行上面的程序，得到以下结果在 /usr/home/log4j/log.out 文件：

```
Warn Message!
Error Message!
Fatal Message!
```

# log4j日志格式化 - Log4j教程

Apache log4j 提供了各种布局对象，每一个对象都可以根据各种布局格式记录数据。另外，也可以创建一个布局对象格式化测井数据中的特定应用的方法。

所有的布局对象 - Appender对象收到 LoggingEvent 对象。布局对象检索来自 LoggingEvent 的消息参数，并应用适当的 ObjectRenderer 获得消息的字符串表示。

## 布局类型：

在层次结构中的顶级类是抽象类是org.apache.log4j.Layout。这是 log4j 的 API 中的所有其他布局类的基类。

布局类定义为抽象在应用程序中，不要直接使用这个类; 相反，使用它的子类来工作，如下：

- DateLayout
- HTMLLayout ( 在本教程解释)
- PatternLayout ( 在本教程解释)
- SimpleLayout
- XMLLayout

## 布局方法：

这个类提供了一个框架实现在所有其它布局对象的所有常见的操作，并声明了两个抽象方法。

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **public abstract boolean ignoresThrowable()** 这种方法表示日志信息是否处理传递给它的日志记录事件的一部分，任何 java.lang.Throwable 对象。如果布局对象处理 Throwable 对象，那么布局对象不忽视它，并返回false。 |
| 2 | **public abstract String format(LoggingEvent event)** 独特的布局子类将实施这一方法的布局特定的格式 |

除了这些抽象方法，布局类提供具体的实现下列方法：

| S.N. | 方法 & 描述 |
|---|---|
| 1 | **public String getContentType()** 返回使用的布局的对象的内容类型。基类将返回 text/plain 作为默认的内容类型 |
| 2 | **public String getFooter()** 指定日志消息的页脚信息 |
| 3 | **public String getHeader()** 指定日志消息的标头信息 |

每个子类可以通过重写的具体实现这些方法返回类特定的信息。

# log4j HTMLLayout - Log4j教程

如果想生成一个HTML格式的文件，日志信息，那么可以使用 org.apache.log4j.HTMLLayout 格式化日志信息。

HTMLLayout类扩展抽象org.apache.log4j.Layout类，并覆盖其基类的 format()方法来提供HTML样式格式。

这提供了以下信息显示：

- 生成特定的日志事件之前，从应用程序的开始所经过的时间
- 调用该记录请求的线程的名称
- 与此记录请求相关联的级别
- 日志记录器(Logger)和记录消息的名称
- 可选程序文件的位置信息，并从其中记录被调用的行号

HTMLLayout是一个非常简单的布局对象，它提供以下方法：

| S.N. | 方法 & 描述 |
| --- | --- |
| 1 | **setContentType(String)** 设置 text/html 为 HTML内容的内容类型。默认为 text/html |
| 2 | **setLocationInfo(String)** 设置位置信息记录事件。默认为 false |
| 3 | **setTitle(String)** 设置为HTML文件的标题。默认值是Log4j的日志信息 |

## HTMLLayout 例子：

以下是对HTMLLayout一个简单的配置文件：

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/htmlLayout.html

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.HTMLLayout
log4j.appender.FILE.layout.Title=HTML Layout Example
log4j.appender.FILE.layout.LocationInfo=true
```

现在考虑下面的Java例子用于产生日志信息：

```
import org.apache.log4j.Logger;

import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class log4jExample{
  /* Get actual class name to be printed on */
  static Logger log = Logger.getLogger(
                      log4jExample.class.getName());

  public static void main(String[] args)
                throws IOException,SQLException{

     log.debug("Hello this is an debug message");
     log.info("Hello this is an info message");
  }
}
```

编译并运行上述程序，它会在 /usr/home/log4j 目录创建 htmlLayout.html 文件，该文件将有如下的日志信息：

Log session start time Mon Mar 22 13:30:24 AST 2014

| Time | Thread | Level | Category | F |
|---|---|---|---|---|
| 0 | main | <font color="#339933">DEBUG</font> | log4jExample | log4jEx |
| 6 | main | INFO | log4jExample | log4jEx |

可以使用一个Web浏览器打开htmlLayout.html 文件。同样重要的是要注意，页脚</HTML>和</ body>标记是完全缺失。

一个具有HTML格式的日志文件的一大优势是，它可以被发布为网页可以远程查看。

# log4j PatternLayout - Log4j教程

如果想生成基于模式的特定格式的日志信息，那么可以使用 org.apache.log4j.PatternLayout 格式化日志信息。

PatternLayout类扩展抽象 org.apache.log4j.Layout 类并覆盖format()方法根据提供的模式构建日志信息。

PatternLayout也是一个简单的布局对象，它提供下列Bean属性，可以通过配置文件进行设置：

| S.N. | 属性和说明 |
|---|---|
| 1 | **conversionPattern** 设置转换模式。默认为 %r [%t] %p %c %x - %m%n |

## 模式转换字符：

下表说明了以上模式使用的字符和所有其他字符，可以在自定义模式中使用：

| 转换字符 | 表示的意思 |
|---|---|
| c | 用于输出的记录事件的类别。例如，对于类别名称"a.b.c" 模式 %c{2} 会输出 "b.c" |
| C | 用于输出呼叫者发出日志请求的完全限定类名。例如，对于类名 "org.apache.xyz.SomeClass", 模式 %C{1} 会输出 "SomeClass". |
| d | 用于输出的记录事件的日期。例如， %d{HH:mm:ss,SSS} 或 %d{dd MMM yyyy HH:mm:ss,SSS}. |
| F | 用于输出被发出日志记录请求，其中的文件名 |
| l | 用于将产生的日志事件调用者输出位置信息 |
| L | 用于输出从被发出日志记录请求的行号 |
| m | 用于输出使用日志事件相关联的应用程序提供的消息 |
| M | 用于输出发出日志请求所在的方法名称 |
| n | 输出平台相关的行分隔符或文字 |
| p | 用于输出的记录事件的优先级 |
| r | 用于输出毫秒从布局的结构经过直到创建日志记录事件的数目 |
| t | 用于输出生成的日志记录事件的线程的名称 |
| x | 用于与产生该日志事件的线程相关联输出的NDC（嵌套诊断上下文） |
| X | 在X转换字符后面是键为的MDC。例如 X{clientIP} 将打印存储在MDC对键clientIP的信息 |
| % | 文字百分号 %%将打印%标志 |

# 格式修饰符：

默认情况下，相关资料原样输出。然而，随着格式修饰符的帮助下，可以改变最小字段宽度，最大字段宽度和对齐。

下表涵盖了各种各样的修饰符的情况：

| Format modifier | left justify | minimum width | maximum width | 注释 |
|---|---|---|---|---|
| %20c | false | 20 | none | 用空格左垫，如果类别名称少于20个字符长 |
| %-20c | true | 20 | none | 用空格右垫，如果类别名称少于20个字符长 |
| %.30c | NA | none | 30 | 从开始截断，如果类别名称超过30个字符长 |
| %20.30c | false | 20 | 30 | 用空格左侧垫，如果类别名称短于20个字符。但是，如果类别名称长度超过30个字符，那么从开始截断。 |
| %-20.30c | true | 20 | 30 | 用空格右侧垫，如果类别名称短于20个字符。但是，如果类别名称长度超过30个字符，那么从开始截断。 |

## PatternLayout 示例:

以下是针对 PatternLayout 一个简单的配置文件：

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=
      %d{yyyy-MM-dd}-%t-%x-%-5p-%-10c:%m%n
```

现在考虑下面产生日志信息的Java例子：

```
import org.apache.log4j.Logger;

import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class log4jExample{
  /* Get actual class name to be printed on */
  static Logger log = Logger.getLogger(
                      log4jExample.class.getName());

  public static void main(String[] args)
                throws IOException,SQLException{

     log.debug("Hello this is an debug message");
     log.info("Hello this is an info message");
  }
}
```

编译并运行上述程序，它会创建 log.out文件在 /usr/home/log4j 目录，该文件将有如下的日志信息：

```
2010-03-23-main--DEBUG-log4jExample:Hello this is an debug message
2010-03-23-main--INFO -log4jExample:Hello this is an info message
```

# log4j日志记录到文件 - Log4j教程

要写日志信息到一个文件中，必须使用org.apache.log4j.FileAppender。有以下FileAppender的配置参数：

## FileAppender配置：

| 属性 | 描述 |
|---|---|
| immediateFlush | 标志的默认设置为true，这意味着输出流的文件被刷新，在每个追加操作 |
| encoding | 它可以使用任何字符编码。默认情况下是特定于平台的编码方案 |
| threshold | 这个 appender 阈值级别 |
| Filename | 日志文件的名称 |
| fileAppend | 默认设置为true，这意味着记录的信息被附加到同一文件的末尾 |
| bufferedIO | 此标志表示是否需要写入缓存启用。默认设置为false |
| bufferSize | 如果 bufferedI/O 启用，这表示缓冲区的大小，默认设置为8KB |

下面是一个示例配置文件 log4j.properties 的 FileAppender。

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
# Set the name of the file
log4j.appender.FILE.File=${log}/log.out

# Set the immediate flush to true (default)
log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode
log4j.appender.FILE.Threshold=debug

# Set the append to false, overwrite
log4j.appender.FILE.Append=false

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

如果喜欢相当于上述log4j.properties文件的XML配置文件，在这里是xml配置文件的内容：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration>

<appender name="FILE" class="org.apache.log4j.FileAppender">
   <param name="file" value="${log}/log.out"/>
   <param name="immediateFlush" value="true"/>
   <param name="threshold" value="debug"/>
   <param name="append" value="false"/>
   <layout class="org.apache.log4j.PatternLayout">
      <param name="conversionPattern" value="%m%n"/>
   </layout>
</appender>

<logger name="log4j.rootLogger" additivity="false">
   <level value="DEBUG"/>
   <appender-ref ref="FILE"/>
</logger>

</log4j:configuration>
```

可以尝试在 log4j - 示例程序 使用上面的配置。

## 日志记录到多个文件：

当想要写日志信息转化多个文件要求一样，例如，如果文件大小达到一定的阈值等。

写日志记录信息分成多个文件，必须扩展FileAppender类，并继承其所有属性useorg.apache.log4j.RollingFileAppender类。

有以下除了已如上所述为 FileAppender 可配置参数：

| 属性 | 描述 |
|---|---|
| maxFileSize | 上述的文件的回滚临界尺寸。默认值是10MB |
| maxBackupIndex | 此属性表示要创建的备份文件的数量。默认值是1 |

下面是一个示例配置文件log4j.properties的RollingFileAppender进行

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.RollingFileAppender
# Set the name of the file
log4j.appender.FILE.File=${log}/log.out

# Set the immediate flush to true (default)
log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode
log4j.appender.FILE.Threshold=debug

# Set the append to false, should not overwrite
log4j.appender.FILE.Append=true

# Set the maximum file size before rollover
log4j.appender.FILE.MaxFileSize=5KB

# Set the the backup index
log4j.appender.FILE.MaxBackupIndex=2

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

如果想有一个XML配置文件，可以生成中提到的初始段，并添加相关的RollingFileAppender 进行唯一额外的参数。

此示例配置说明每个日志文件的最大允许大小为5MB。当超过最大尺寸，新的日志文件将被创建并因为maxBackupIndex被定义为2，当第二个日志文件达到最大值，第一个日志文件将被删除，之后所有的日志信息将被回滚到第一个日志文件。

可以尝试 log4j - 示例程序使用上面的配置。

# 每天生成日志文件：

当想生成每一天的日志文件，以保持日志记录信息的良好记录。

日志记录信息纳入日常的基础文件，就必须它扩展FileAppender类，并继承其所有属性useorg.apache.log4j.DailyRollingFileAppender类。

有除了已如上所述为 FileAppender 只有一个重要的下列配置参数：

| Property | 描述 |
|---|---|
| DatePattern | 这表示在滚动的文件，并按命名惯例来执行。默认情况下，在每天午夜滚动 |

DatePattern控制使用下列滚动的时间表方式之一：

| DatePattern | 描述 |
|---|---|
| '.' yyyy-MM | 滚动在每个月的结束和下一个月初 |
| '.' yyyy-MM-dd | 这是默认值，每天午夜滚动 |
| '.' yyyy-MM-dd-a | 滚动每一天的午夜和中午 |
| '.' yyyy-MM-dd-HH | 滚动在每一个小时 |
| '.' yyyy-MM-dd-HH-mm | 滚动在每一个分钟 |
| '.' yyyy-ww | 滚动每个星期取决于区域设置时的第一天 |

下面是一个示例配置文件log4j.properties生成日志文件滚动的在每天午夜。

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.DailyRollingFileAppender
# Set the name of the file
log4j.appender.FILE.File=${log}/log.out

# Set the immediate flush to true (default)
log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode
log4j.appender.FILE.Threshold=debug

# Set the append to false, should not overwrite
log4j.appender.FILE.Append=true

# Set the DatePattern
log4j.appender.FILE.DatePattern='.' yyyy-MM-dd-a

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

如果想使用XML配置文件，可以生成中提到的初始段，并添加相关DailyRollingFileAppender 唯一的额外参数和数据。

可以尝试在 log4j - 示例程序使用上面的配置。

# log4j日志记录到数据库 - Log4j教程

log4j API提供 org.apache.log4j.jdbc.JDBCAppender 对象，它能够将日志信息在指定的数据库。

## JDBCAppender 配置:

| Property | 描述 |
|---|---|
| bufferSize | 设置缓冲区的大小。默认大小为1 |
| driver | 设置驱动程序类为指定的字符串。如果没有指定驱动程序类，默认为sun.jdbc.odbc.JdbcOdbcDriver |
| layout | 设置要使用的布局。默认布局是org.apache.log4j.PatternLayout |
| password | Sets the database password. |
| sql | 指定SQL语句在每次记录事件发生的时间执行。这可能是INSERT，UPDATE或DELETE |
| URL | 设置JDBC URL |
| user | 设置数据库用户名 |

## 日志表配置：

开始使用基于JDBC日志，要创建在哪里保存日志信息的表。下面是创建日志表的SQL语句：

```
CREATE TABLE LOGS
   (USER_ID VARCHAR(20) NOT NULL,
    DATED   DATE NOT NULL,
    LOGGER  VARCHAR(50) NOT NULL,
    LEVEL   VARCHAR(10) NOT NULL,
    MESSAGE VARCHAR(1000) NOT NULL
   );
```

## 配置文件示例：

以下是将用于将消息记录到一个日志表中的示例配置文件 log4j.properties的JDBCAppender

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, DB

# Define the DB appender
log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender

# Set JDBC URL
log4j.appender.DB.URL=jdbc:mysql://localhost/DBNAME

# Set Database Driver
log4j.appender.DB.driver=com.mysql.jdbc.Driver

# Set database user name and password
log4j.appender.DB.user=user_name
log4j.appender.DB.password=password

# Set the SQL statement to be executed.
log4j.appender.DB.sql=INSERT INTO LOGS
                      VALUES('%x','%d','%C','%p','%m')

# Define the layout for file appender
log4j.appender.DB.layout=org.apache.log4j.PatternLayout
```

这里使用的是MySQL数据库，必须要使用实际DBNAME，用户ID和在其中创建的日志表的数据库密码。SQL语句是使用日志表名和输入值到表，需要执行INSERT语句。

JDBCAppender不需要明确定义的布局。相反，使用PatternLayout 传递给它 SQL语句

如果想拥有相当于上述log4j.properties文件的XML配置文件，可以参考在这里的内容：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration>

<appender name="DB" class="org.apache.log4j.jdbc.JDBCAppender">
   <param name="url" value="jdbc:mysql://localhost/DBNAME"/>
   <param name="driver" value="com.mysql.jdbc.Driver"/>
   <param name="user" value="user_id"/>
   <param name="password" value="password"/>
   <param name="sql" value="INSERT INTO LOGS VALUES('%x',
                             '%d','%C','%p','%m')"/>
   <layout class="org.apache.log4j.PatternLayout">
   </layout>

</appender>

<logger name="log4j.rootLogger" additivity="false">
   <level value="DEBUG"/>
   <appender-ref ref="DB"/>
</logger>

</log4j:configuration>
```

## 示例程序：

下面的Java类是一个非常简单的Java应用程序使用Log4J日志库例子，初始化，然后使用。

```
import org.apache.log4j.Logger;
import java.sql.*;
import java.io.*;
import java.util.*;

public class log4jExample{
  /* Get actual class name to be printed on */
  static Logger log = Logger.getLogger(
                      log4jExample.class.getName());

  public static void main(String[] args)
                throws IOException,SQLException{

     log.debug("Debug");
     log.info("Info");
  }
}
```

## 编译和运行：

下面是步骤编译并运行上述程序。确保进行编译和执行之前，适当地设置PATH和CLASSPATH。

所有的库应该在CLASSPATH以及log4j.properties文件应该在PATH可用。所以有以下几点：

- 创建log4j.properties如上图所示。
- 创建log4jExample.java如上图所示，并对其进行编译。
- 执行log4jExample二进制运行程序。

现在检查DBNAME数据库里面日志表，发现下面的条目（记录）：

```
mysql >  select * from LOGS;
+---------+------------+--------------+-------+---------+
| USER_ID | DATED      | LOGGER       | LEVEL | MESSAGE |
+---------+------------+--------------+-------+---------+
|         | 2010-05-13 | log4jExample | DEBUG | Debug   |
|         | 2010-05-13 | log4jExample | INFO  | Info    |
+---------+------------+--------------+-------+---------+
2 rows in set (0.00 sec)
```

注：此处X被用于产生该记录事件的线程相关联输出的NDC（嵌套诊断上下文）。使用NDC来区分客户的服务器端组件处理多个客户端。检查Log4J的手册以获取更多信息。

# Lucene教程

Lucene是简单而功能强大的基于Java的搜索库。它可以用于任何应用程序来搜索功能。 Lucene是开源项目。它是可扩展的，高性能的库用于索引和搜索几乎任何类型的文本。 Lucene库提供了所需的任何搜索应用程序的核心业务。索引和搜索。

## 搜索应用程序的工作原理

任何搜索应用程序执行一些或全部下列操作。

| 步骤 | 标题 | 描述 |
| --- | --- | --- |
| 1 | 获取原始内容 | 任何搜索应用程序的第一个步骤是收集在其上的搜索是要进行的目标内容。 |
| 2 | 构建文档 | 下一步是建立从原始内容的搜索应用程序可以理解和容易理解的文件。 |
| 3 | 分析文档 | 在索引过程启动，该文件是要分析作为其文本部分是一个候选索引。这个过程被称为分析文档。 |
| 4 | 索引文件 | 一旦文档被构建和分析，下一步是将索引它们使得该文件可被检索 |

# Lucene环境设置 - Lucene教程

## 环境设置

本教程将指导如何准备一个开发环境，开始与Spring框架工作。本教程还将教如何安装JDK，Tomcat和Eclipse在机器上在设置Spring框架之前：

## 第1步 - 安装Java开发工具包(JDK)：

可以从OracleJava网站上找到最新版本的SDK：Java SE下载。发现有说明在下载的文件中说明处理安装JDK，按照说明安装和配置设置的指示。最后，设置PATH和JAVA_HOME环境变量指的是分别包含java和javac，通常java_install_dir/bin和java_install_dir目录。

如果运行的是Windows，并安装了JDK在 C:jdk1.6.0_15，就必须执行以下C:autoexec.bat。

```
set PATH=C:jdk1.6.0_15in;%PATH%
set JAVA_HOME=C:jdk1.6.0_15
```

另外如果在Windows NT/2000/XP中，也可以在“我的电脑”，选择“属性”，然后“高级”，然后环境变量单击鼠标右键。更新PATH的值，然后按OK按钮。

在Unix（Solaris和Linux等），如果SDK安装在/usr/local/jdk1.6.0_15并且使用的是C shell，把下列内容放入.cshrc文件。

```
setenv PATH /usr/local/jdk1.6.0_15/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.6.0_15
```

另外，如果使用的集成开发环境（IDE）像Borland公司的JBuilder，Eclipse或IntelliJ IDEA或Sun ONE Studio，那么编译并运行一个简单的程序，以确认该IDE知道安装Java，否则需要为IDE做适当的设置。

## 第2步 - 安装Eclipse IDE

在本教程中的所有示例使用Eclipse IDE。所以建议你应该在计算机上安装最新版本的 Eclipse。

要安装Eclipse IDE，请从 http://www.eclipse.org/downloads/ 上下载最新的 Eclipse可执行文件并安装，解压缩二进制分发到一个方便的位置。例如，在C：Eclipse在Windows上，或在Linux/Unix的 /usr/local/eclipse，最后设置PATH变量恰当即可。

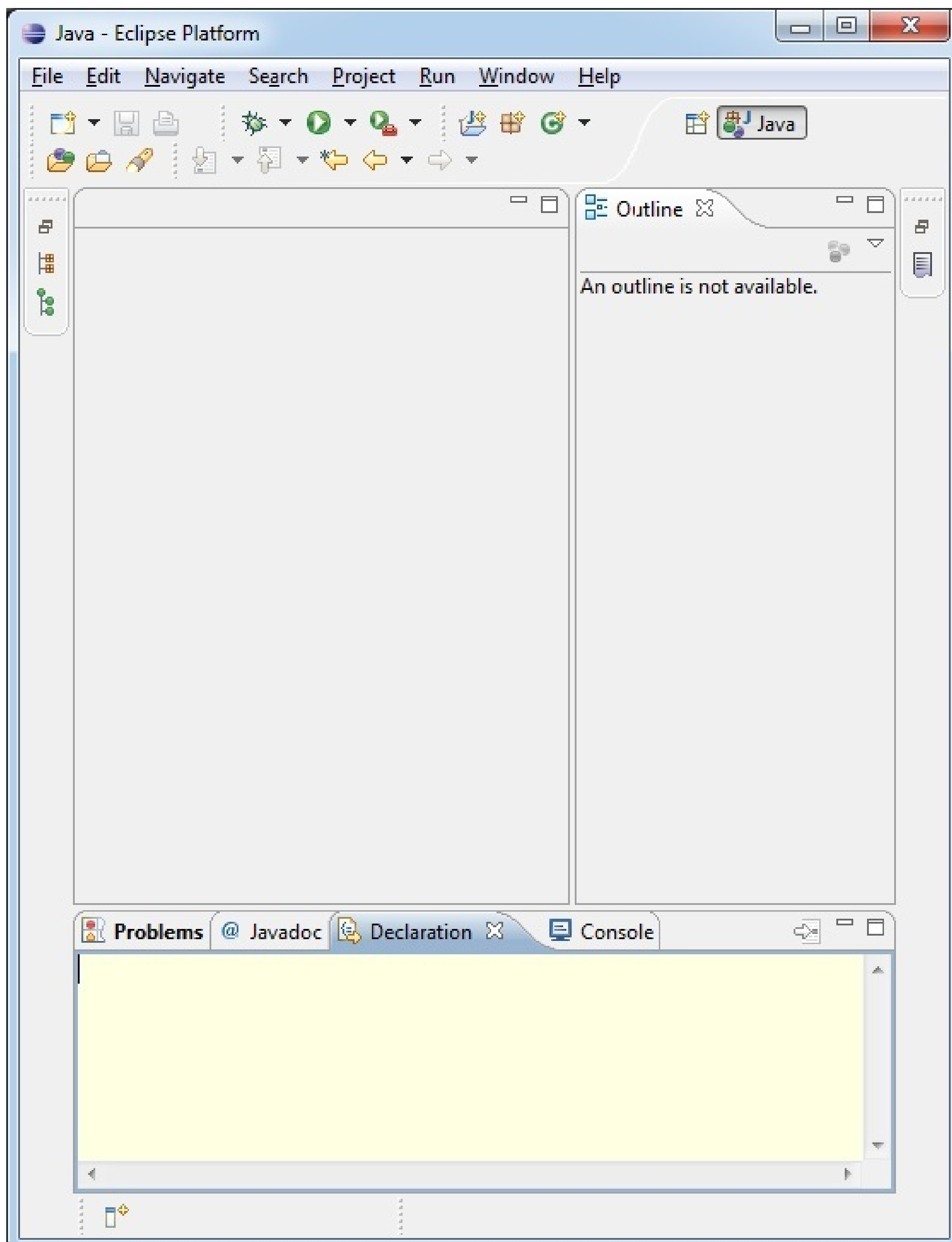
Eclipse可以通过在Windows计算机上执行以下命令来启动，或者可以简单地双击eclipse.exe

```
%C:eclipseeclipse.exe
```

Eclipse可以通过执行在Unix（Solaris和Linux等）机器下面的命令来启动：

```
$/usr/local/eclipse/eclipse
```

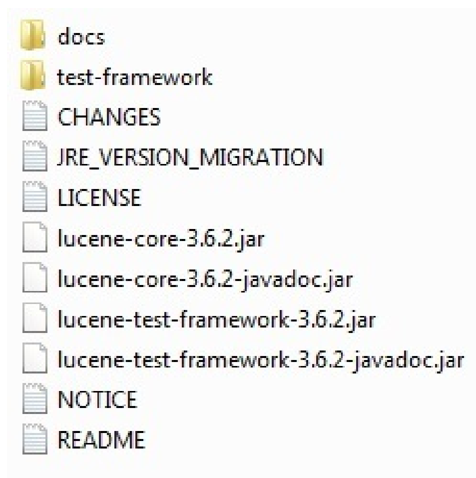
成功启动后，如果一切正常那么就应该显示如下结果：



# 第3步 - 安装Lucene框架库

现在，如果一切正常，那么就可以继续设的Lucene框架。以下是简单的步骤下载并在机器上安装Lucene框架。

- 选择是否要在Windows或UNIX上安装Lucene，然后进行下一个步骤下载 .zip 文件适用于Windows，以及 .tz 文件适用在 Unix 中。
- 从下载合适版本的Lucene的框架二进制文件 http://archive.apache.org/dist/lucene/java/.
- 在写这篇教程的时候，下载lucene-3.6.2.zip在Windows机器上，当解压缩下载的文件，它会给出里面的目录结构在 C:lucene-3.6.2 如下。




会发现所有的Lucene库在C:lucene-3.6.2 目录。确保在这个目录上设置CLASSPATH变量正确，否则会在运行应用程序时出现问题。如果使用的是 Eclipse 则不需要设置 CLASSPATH，因为所有的设置已通过Eclipse进行。

最后一步完成后，就可以开始Lucene的第一个例子，在下一章中将看到。

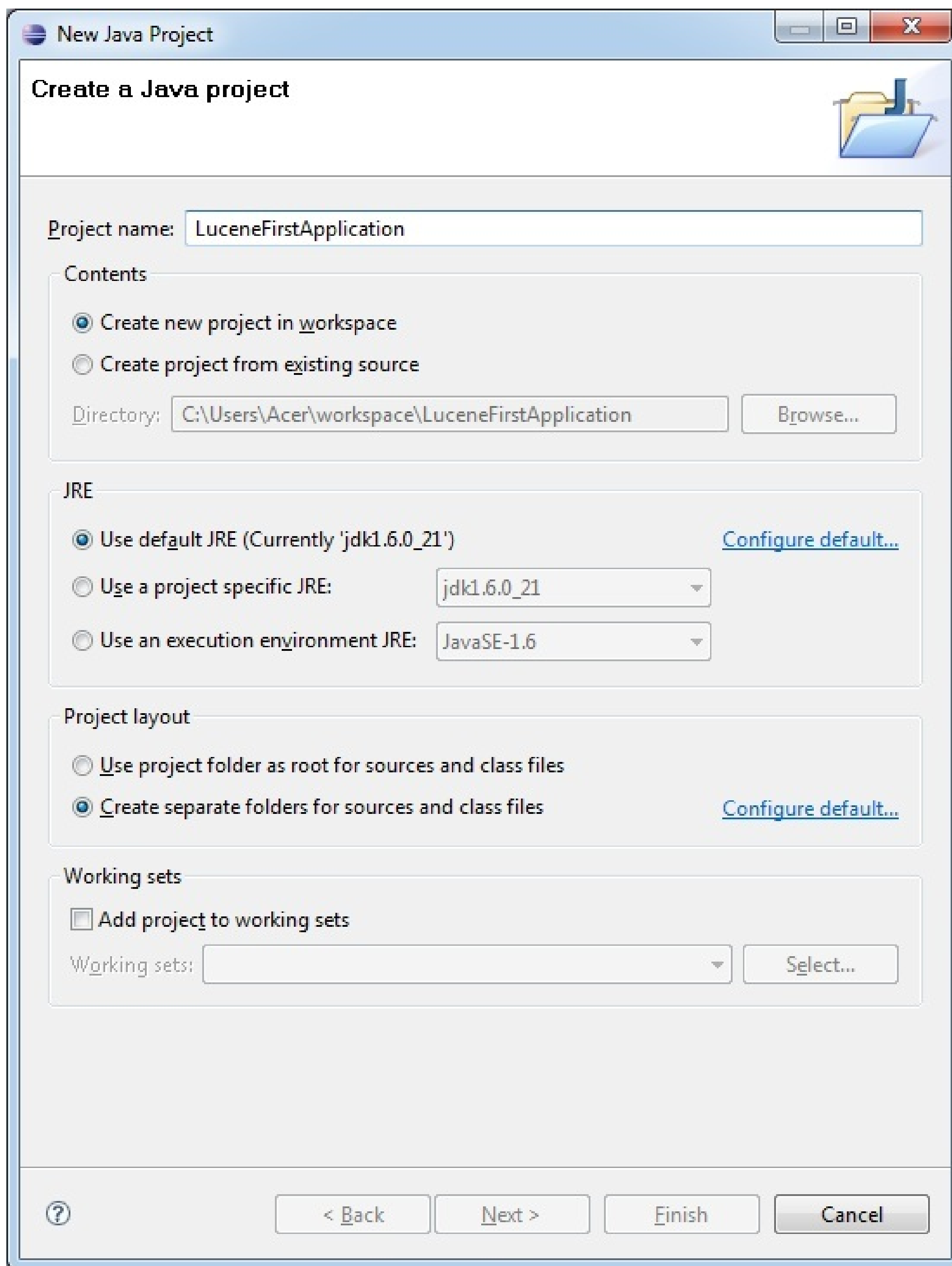
# Lucene第一个应用程序 - Lucene教程

让我们使用Lucene框架做实际编程。在开始使用Lucene框架编写第一个例子之前，必须确保已经安装Lucene的环境正常。也假设有一点点的工作和Eclipse IDE的知识。

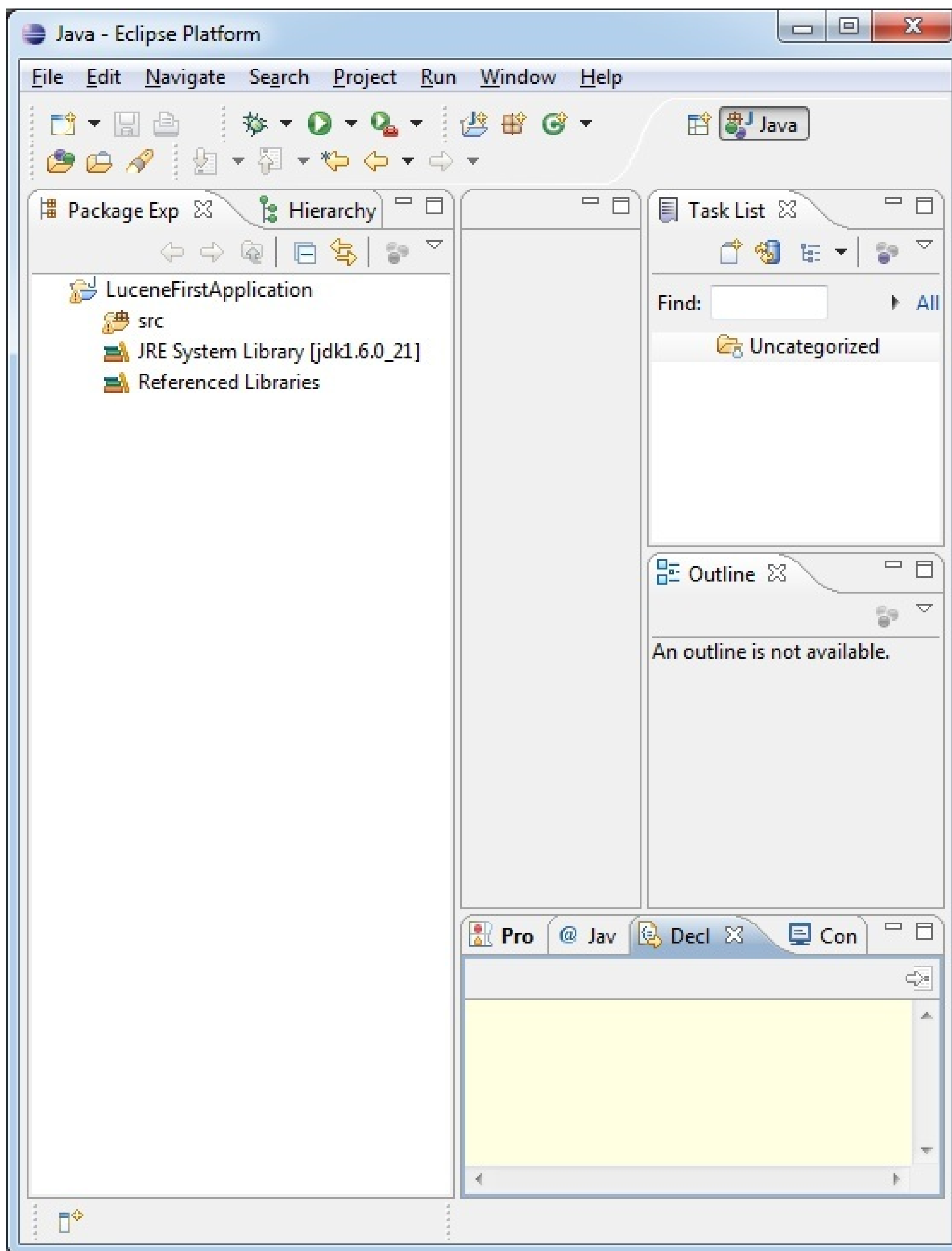
因此，开始写一个简单的搜索应用程序将打印找到搜索结果数量。我们也看到在这个过程中创建的索引列表。

## 第1步 - 创建Java项目：

第一步是使用**Eclipse IDE**创建一个简单的Java项目。按照选项 **File -> New -> Project** 最后选择 **Java Project** 从向导列表向导。现在，项目命名为 **LuceneFirstApplication** 使用向导窗口，如下所示：

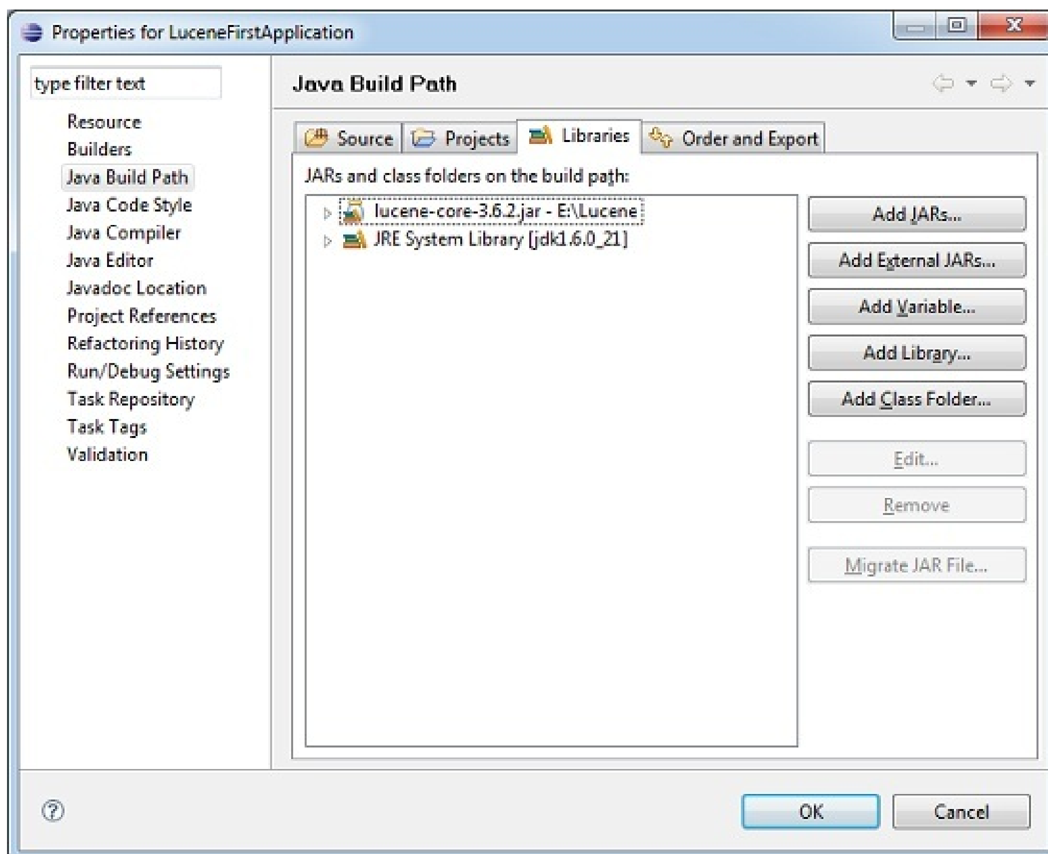



一旦项目成功创建，将有以下内容在 **Project Explorer:**




## 第2步 - 添加必需的库 ：

作为第二步，我们添加Lucene核心框架库在项目中。要做到这一点，右键单击项目名称LuceneFirstApplication然后按照上下文菜单中提供以下选项：Build Path -> Configure Build Path，显示了Java构建路径如下窗口：




现在，使用添加在库选项卡中提供外部JAR按钮，添加Lucene安装目录下的核心JAR：

- lucene-core-3.6.2

# 第3步 - 创建源文件：

现在，让我们 **LuceneFirstApplication** 项目下创建实际的源文件。首先，我们需要创建一个名为 **com.yiibai.lucene** 包。要做到这一点，右键单击 src 在包资源管理部分，并按照选项：**New -> Package.**

下一步，我们将创建 LuceneTester.java 和 其他Java类在 **com.yiibai.lucene** 包下。

### LuceneConstants.java

这个类是用来提供跨示例应用程序中使用的各种常量。

```
package com.yiibai.lucene;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

*TextFileFilter.java*

此类用于为 .txt 文件过滤器

```
package com.yiibai.lucene;

import java.io.File;
import java.io.FileFilter;

public class TextFileFilter implements FileFilter {

   @Override
   public boolean accept(File pathname) {
      return pathname.getName().toLowerCase().endsWith(".txt");
   }
}
```

*Indexer.java*

这个类是用于索引的原始数据，这样我们就可以使用Lucene库，使其可搜索。

```
package com.yiibai.lucene;

import java.io.File;
import java.io.FileFilter;
import java.io.FileReader;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Indexer {

   private IndexWriter writer;

   public Indexer(String indexDirectoryPath) throws IOException{
      //this directory will contain the indexes
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));

      //create the indexer
      writer = new IndexWriter(indexDirectory,
         new StandardAnalyzer(Version.LUCENE_36),true,
         IndexWriter.MaxFieldLength.UNLIMITED);
   }
```

```
    public void close() throws CorruptIndexException, IOException{
       writer.close();
    }

    private Document getDocument(File file) throws IOException{
       Document document = new Document();

       //index file contents
       Field contentField = new Field(LuceneConstants.CONTENTS,
          new FileReader(file));
       //index file name
       Field fileNameField = new Field(LuceneConstants.FILE_NAME,
          file.getName(),
          Field.Store.YES,Field.Index.NOT_ANALYZED);
       //index file path
       Field filePathField = new Field(LuceneConstants.FILE_PATH,
          file.getCanonicalPath(),
          Field.Store.YES,Field.Index.NOT_ANALYZED);

       document.add(contentField);
       document.add(fileNameField);
       document.add(filePathField);

       return document;
    }

    private void indexFile(File file) throws IOException{
       System.out.println("Indexing "+file.getCanonicalPath());
       Document document = getDocument(file);
       writer.addDocument(document);
    }

    public int createIndex(String dataDirPath, FileFilter filter)
       throws IOException{
       //get all files in the data directory
       File[] files = new File(dataDirPath).listFiles();

       for (File file : files) {
          if(!file.isDirectory()
             && !file.isHidden()
             && file.exists()
             && file.canRead()
             && filter.accept(file)
          ){
             indexFile(file);
          }
       }
       return writer.numDocs();
    }
}
```

*Searcher.java*

这个类是用来搜索索引所创建的索引搜索请求的内容。

```
package com.yiibai.lucene;

import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Searcher {

   IndexSearcher indexSearcher;
   QueryParser queryParser;
   Query query;

   public Searcher(String indexDirectoryPath)
      throws IOException{
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));
      indexSearcher = new IndexSearcher(indexDirectory);
      queryParser = new QueryParser(Version.LUCENE_36,
         LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
   }

   public TopDocs search( String searchQuery)
      throws IOException, ParseException{
      query = queryParser.parse(searchQuery);
      return indexSearcher.search(query, LuceneConstants.MAX_SEARCH
   }

   public Document getDocument(ScoreDoc scoreDoc)
      throws CorruptIndexException, IOException{
      return indexSearcher.doc(scoreDoc.doc);
   }

   public void close() throws IOException{
      indexSearcher.close();
   }
}
```

*LuceneTester.java*

这个类是用来测试 Lucene 库的索引和搜索功能。

```
package com.yiibai.lucene;

import java.io.IOException;

import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;

public class LuceneTester {

   String indexDir = "E:\Lucene\Index";
   String dataDir = "E:\Lucene\Data";
   Indexer indexer;
   Searcher searcher;

   public static void main(String[] args) {
      LuceneTester tester;
      try {
         tester = new LuceneTester();
         tester.createIndex();
         tester.search("Mohan");
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }

   private void createIndex() throws IOException{
      indexer = new Indexer(indexDir);
      int numIndexed;
      long startTime = System.currentTimeMillis();
      numIndexed = indexer.createIndex(dataDir, new TextFileFilter(
      long endTime = System.currentTimeMillis();
      indexer.close();
      System.out.println(numIndexed+" File indexed, time taken: "
         +(endTime-startTime)+" ms");
   }

   private void search(String searchQuery) throws IOException, Pars
      searcher = new Searcher(indexDir);
      long startTime = System.currentTimeMillis();
      TopDocs hits = searcher.search(searchQuery);
      long endTime = System.currentTimeMillis();

      System.out.println(hits.totalHits +
         " documents found. Time :" + (endTime - startTime));
```

```
        for(ScoreDoc scoreDoc : hits.scoreDocs) {
           Document doc = searcher.getDocument(scoreDoc);
              System.out.println("File: "
              + doc.get(LuceneConstants.FILE_PATH));
        }
        searcher.close();
     }
}
```

## 第4步- 创建数据和索引目录

把 record1.txt 任命为 record10.txt 包含简单的名称以及学生的其他数据信息，并把它们放在目录：E:LuceneData 并测试数据。索引目录路径应创建在 E:LuceneIndex。运行此程序后，就可以看到该文件夹中创建的索引文件的列表。

## 第5步 - 运行程序：

一旦完成创建源和原始数据，数据目录和索引目录，下一步是编译和运行程序。要做到这一点，请LuceneTester.Java文件的活动选项卡中使用EclipseIDE可无论是运行选项，或使用Ctrl+ F11来编译和运行应用程序LuceneTester。如果一切正常您的应用程序，这将打印在 Eclipse IDE 控制台以下消息：

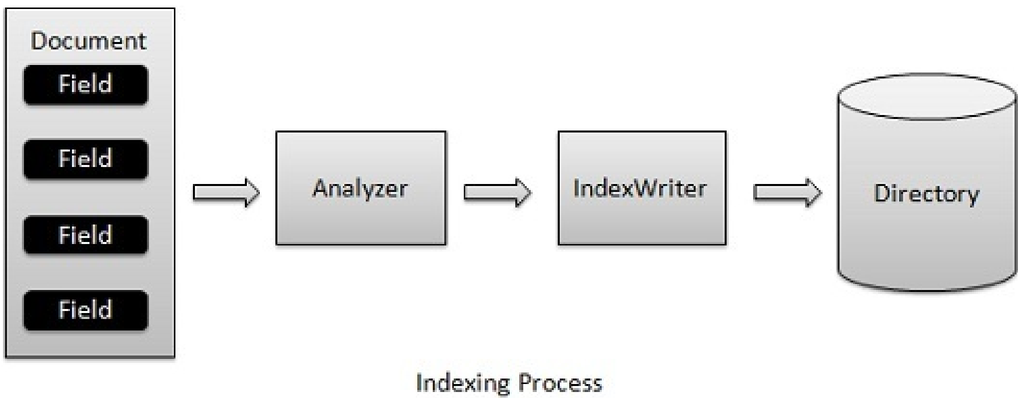
```
Indexing E:LuceneData
ecord1.txt
Indexing E:LuceneData
ecord10.txt
Indexing E:LuceneData
ecord2.txt
Indexing E:LuceneData
ecord3.txt
Indexing E:LuceneData
ecord4.txt
Indexing E:LuceneData
ecord5.txt
Indexing E:LuceneData
ecord6.txt
Indexing E:LuceneData
ecord7.txt
Indexing E:LuceneData
ecord8.txt
Indexing E:LuceneData
ecord9.txt
10 File indexed, time taken: 109 ms
1 documents found. Time :0
File: E:LuceneData
ecord4.txt
```

一旦已经成功地运行程序，将有以下的索引目录中的内容：

| Name | Date modified | Type | Size |
|---|---|---|---|
| _0.fdt | 5/25/2014 3:15 PM | FDT File | 1 KB |
| _0.fdx | 5/25/2014 3:15 PM | FDX File | 1 KB |
| _0.fnm | 5/25/2014 3:15 PM | FNM File | 1 KB |
| _0.frq | 5/25/2014 3:15 PM | FRQ File | 1 KB |
| _0 | 5/25/2014 3:15 PM | Mixed Mode CD C... | 1 KB |
| _0.prx | 5/25/2014 3:15 PM | PRX File | 1 KB |
| _0.tii | 5/25/2014 3:15 PM | TII File | 1 KB |
| _0.tis | 5/25/2014 3:15 PM | TIS File | 1 KB |
| segments.gen | 5/25/2014 3:15 PM | GEN File | 1 KB |
| segments_1 | 5/25/2014 3:15 PM | File | 1 KB |

# Lucene索引类 - Lucene教程

索引过程是由Lucene所提供的核心功能之一。下图说明了索引过程和使用的类。IndexWriter 是索引过程中最重要的和核心组件。




Indexing Process

我们添加包含字段的 IndexWriter 分析使用 Analyzer 文件，然后创建/根据需要打开/编辑索引和存储/在目录更新。IndexWriter用于更新或创建索引。它不是用来读取索引。

## Indexing类：

以下是常用索引进程类的列表。

| Sr. No. | 类及描述 |
|---|---|
| 1 | IndexWriter 此类充当创造/在索引过程中更新指标的核心组成部分 |
| 2 | Directory 此类表示索引的存储位置 |
| 3 | Analyzer Analyzer类负责分析一个文件，并从将被索引的文本获取令牌/字。不加分析完成后，IndexWriter不能创建索引。 |
| 4 | Document Document代表一个虚拟文档与字段，其中字段是可包含在物理文档的内容，元数据等对象。Analyzer只能理解文档。 |
| 5 | Field Field是最低单元或索引过程的起点。它代表其中一个键被用于识别要被索引的值的键值对关系。用于表示一个文件内容的字段将具有键为“内容”，值可以包含文本或文档的数字内容的部分或全部。 Lucene能索引仅文本或仅数字内容。 |

# Lucene Searching类 - Lucene教程

在搜索过程是由Lucene所提供的核心功能之一。它的流程是相似于索引过程。Lucene基本搜索可以使用下列类也可称为基础类的所有搜索相关的操作进行。

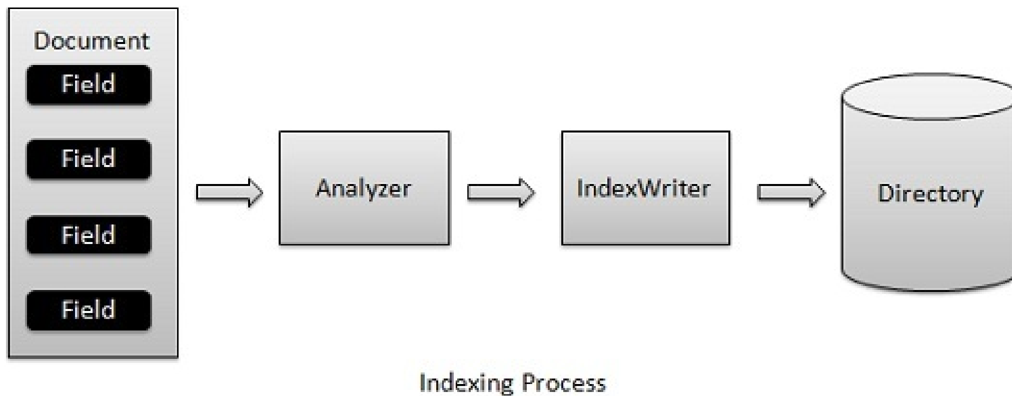
## Searching 类：

以下是常用的类中搜索处理的列表。

| Sr. No. | 类和说明 |
| --- | --- |
| 1 | IndexSearcher 这个类充当读取/搜索索引的过程后创建索引的核心组成部分。它需要目录实例指向包含索引的位置 |
| 2 | Term 这个类是搜索的最低单位。它是在索引过程中类似字段 Field |
| 3 | Query Query是一个抽象类，包含各种实用方法，所有类型查询的父在Lucene的搜索过程中使用 |
| 4 | TermQuery TermQuery是最常用的查询对象，并且是许多复杂的查询lucene可利用的基础 |
| 5 | TopDocs TopDocs指向相匹配的搜索条件的前N个搜索结果。它是指针的简单容器指向它们的搜索结果输出的文档。 |

# Lucene索引过程 - Lucene教程

索引过程是Lucene提供的核心功能之一。下图说明了索引过程和使用的类。IndexWriter是索引过程中最重要的和核心组件。




Indexing Process

添加文档包含字段IndexWriter，该分析用分析仪分析文件，然后创建/根据需要并在目录存储/更新/打开/编辑索引。IndexWriter用于更新或创建索引。它不是用来读取索引。

现在，展示一个循序渐进的过程，以获得在索引过程的理解，使用一个基本的例子。

## 创建一个文档

- 创建一个方法来获取从文本文件中获得 Lucene 的文档。
- 创建各种类型的是含有键作为名称和值作为内容被编入索引键值对字段。
- 设置字段中进行分析或不设置。在我们的实例中，只有内容被分析，因为它可能包含数据，诸如 a, am, are, an，它不要求在搜索操作等等。
- 新创建的字段添加到文档对象并返回给调用者的方法。

```
private Document getDocument(File file) throws IOException{
   Document document = new Document();

   //index file contents
   Field contentField = new Field(LuceneConstants.CONTENTS,
      new FileReader(file));
   //index file name
   Field fileNameField = new Field(LuceneConstants.FILE_NAME,
      file.getName(),
      Field.Store.YES,Field.Index.NOT_ANALYZED);
   //index file path
   Field filePathField = new Field(LuceneConstants.FILE_PATH,
      file.getCanonicalPath(),
      Field.Store.YES,Field.Index.NOT_ANALYZED);

   document.add(contentField);
   document.add(fileNameField);
   document.add(filePathField);

   return document;
}
```

## 创建IndexWriter

- IndexWriter 类作为它创建/在索引过程中更新指标的核心组成部分
- 创建一个 IndexWriter 对象
- 创建其应指向位置，其中索引是存储一个lucene的目录
- 初始化索引目录，有标准的分析版本信息和其他所需/可选参数创建 IndexWriter 对象

```
private IndexWriter writer;

public Indexer(String indexDirectoryPath) throws IOException{
   //this directory will contain the indexes
   Directory indexDirectory =
      FSDirectory.open(new File(indexDirectoryPath));
   //create the indexer
   writer = new IndexWriter(indexDirectory,
      new StandardAnalyzer(Version.LUCENE_36),true,
      IndexWriter.MaxFieldLength.UNLIMITED);
}
```

## 开始索引过程

```
private void indexFile(File file) throws IOException{
   System.out.println("Indexing "+file.getCanonicalPath());
   Document document = getDocument(file);
   writer.addDocument(document);
}
```

# 应用程序示例

让我们创建一个测试 Lucene 应用程序来测试索引过程。

| 步骤 | 描述 |
|---|---|
| 1 | 在 packagecom.yiibai.lucene 包下创建一个名称 LuceneFirstApplication 项目用于解释 Lucene - First Application chapter, 也可以使用 Lucene 的创建项目 - 在 First Application 章这样本章理解索引过程。 |
| 2 | 创建LuceneConstants.java， TextFileFilter.java和 Indexer.java，其它的文件保存不变。 |
| 3 | 创建LuceneTester.java如下所述 |
| 4 | 清理和构建应用程序，以确保业务逻辑按要求 |

*LuceneConstants.java*

这个类是用来提供跨示例应用程序中使用的各种常量

```
package com.yiibai.lucene;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

*TextFileFilter.java*

此类用于为 .txt 文件过滤器

```
package com.yiibai.lucene;

import java.io.File;
import java.io.FileFilter;

public class TextFileFilter implements FileFilter {

   @Override
   public boolean accept(File pathname) {
      return pathname.getName().toLowerCase().endsWith(".txt");
   }
}
```

*Indexer.java*

这个类是用于索引的原始数据，这样就可以使用Lucene库，使其可搜索。

```
package com.yiibai.lucene;

import java.io.File;
import java.io.FileFilter;
import java.io.FileReader;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Indexer {

   private IndexWriter writer;

   public Indexer(String indexDirectoryPath) throws IOException{
      //this directory will contain the indexes
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));

      //create the indexer
      writer = new IndexWriter(indexDirectory,
         new StandardAnalyzer(Version.LUCENE_36),true,
         IndexWriter.MaxFieldLength.UNLIMITED);
   }

   public void close() throws CorruptIndexException, IOException{
      writer.close();
   }
```

```
   private Document getDocument(File file) throws IOException{
      Document document = new Document();

      //index file contents
      Field contentField = new Field(LuceneConstants.CONTENTS,
         new FileReader(file));
      //index file name
      Field fileNameField = new Field(LuceneConstants.FILE_NAME,
         file.getName(),
         Field.Store.YES,Field.Index.NOT_ANALYZED);
      //index file path
      Field filePathField = new Field(LuceneConstants.FILE_PATH,
         file.getCanonicalPath(),
         Field.Store.YES,Field.Index.NOT_ANALYZED);

      document.add(contentField);
      document.add(fileNameField);
      document.add(filePathField);

      return document;
   }

   private void indexFile(File file) throws IOException{
      System.out.println("Indexing "+file.getCanonicalPath());
      Document document = getDocument(file);
      writer.addDocument(document);
   }

   public int createIndex(String dataDirPath, FileFilter filter)
      throws IOException{
      //get all files in the data directory
      File[] files = new File(dataDirPath).listFiles();

      for (File file : files) {
         if(!file.isDirectory()
            && !file.isHidden()
            && file.exists()
            && file.canRead()
            && filter.accept(file)
         ){
            indexFile(file);
         }
      }
      return writer.numDocs();
   }
}
```

*LuceneTester.java*

这个类是用来测试 Lucene 库的索引能力。

```
package com.yiibai.lucene;

import java.io.IOException;

public class LuceneTester {

   String indexDir = "E:\Lucene\Index";
   String dataDir = "E:\Lucene\Data";
   Indexer indexer;

   public static void main(String[] args) {
      LuceneTester tester;
      try {
         tester = new LuceneTester();
         tester.createIndex();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   private void createIndex() throws IOException{
      indexer = new Indexer(indexDir);
      int numIndexed;
      long startTime = System.currentTimeMillis();
      numIndexed = indexer.createIndex(dataDir, new TextFileFilter(
      long endTime = System.currentTimeMillis();
      indexer.close();
      System.out.println(numIndexed+" File indexed, time taken: "
         +(endTime-startTime)+" ms");
   }
}
```

## 数据和索引目录的创建

从 record1.txt 命名文件 record10.txt 包含简单的名称以及学生的其他细节，并把它们放在目录 E:LuceneData. 索引目录路径应创建为 E:LuceneIndex. 运行此程序后，就可以看到该文件夹中创建的索引文件的列表。

## 运行程序：

一旦使用创建源，创造了原始数据，数据目录和索引目录来完成，准备好这一步然后编译和运行程序。要做到这一点，保存LuceneTester.Java文件选项卡中使用Eclipse IDE 运行 Run 选项，或使用Ctrl+ F11来编译和运行应用程序LuceneTester。如果应用程序一切正常，这将打印在Eclipse IDE的控制台以下消息：

```
Indexing E:LuceneData
ecord1.txt
Indexing E:LuceneData
ecord10.txt
Indexing E:LuceneData
ecord2.txt
Indexing E:LuceneData
ecord3.txt
Indexing E:LuceneData
ecord4.txt
Indexing E:LuceneData
ecord5.txt
Indexing E:LuceneData
ecord6.txt
Indexing E:LuceneData
ecord7.txt
Indexing E:LuceneData
ecord8.txt
Indexing E:LuceneData
ecord9.txt
10 File indexed, time taken: 109 ms
```

一旦成功地运行程序，将有以下的索引目录中的内容：

| Name | Date modified | Type | Size |
|---|---|---|---|
| _0.fdt | 5/25/2014 3:15 PM | FDT File | 1 KB |
| _0.fdx | 5/25/2014 3:15 PM | FDX File | 1 KB |
| _0.fnm | 5/25/2014 3:15 PM | FNM File | 1 KB |
| _0.frq | 5/25/2014 3:15 PM | FRQ File | 1 KB |
| _0 | 5/25/2014 3:15 PM | Mixed Mode CD C... | 1 KB |
| _0.prx | 5/25/2014 3:15 PM | PRX File | 1 KB |
| _0.tii | 5/25/2014 3:15 PM | TII File | 1 KB |
| _0.tis | 5/25/2014 3:15 PM | TIS File | 1 KB |
| segments.gen | 5/25/2014 3:15 PM | GEN File | 1 KB |
| segments_1 | 5/25/2014 3:15 PM | File | 1 KB |

# Lucene索引操作 - Lucene教程

在本章中，我们将讨论索引的四个主要操作。这些操作是在不同的时间和有用用于整个软件搜索应用程序。

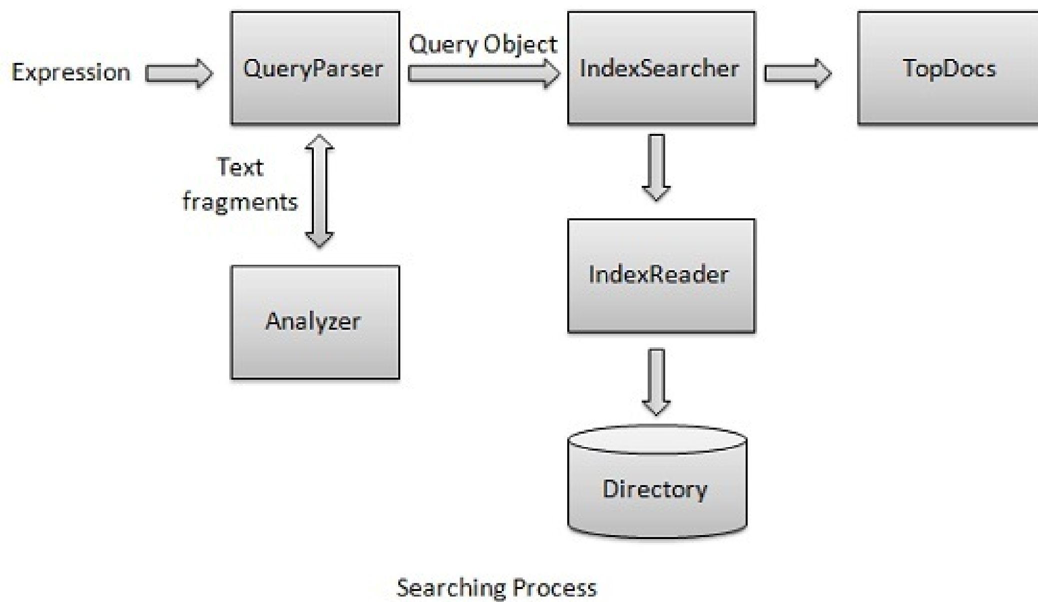
## 索引操作：

以下是常用的操作过程中索引进程列表。

| Sr. No. | 操作说明： |
|---|---|
| 1 | Add Document 此操作在索引进程列表的初始阶段用于在新的可用内容创建索引 |
| 2 | Update Document 此操作用于更新索引以反映变化的更新的内容。它类似于重新创建索引 |
| 3 | Delete Document 此操作用于更新的索引以排除它们不需要被索引的文档/搜索 |
| 4 | Field Options 字段选项指定的方式或控制的方式，一个字段的内容进行搜索 |

# Lucene搜索操作 - Lucene教程

搜索过程是由Lucene所提供的核心功能之一。下图说明了搜索过程和使用的类。IndexSearcher是搜索过程中最重要的和核心组件。




Searching Process

我们首先创建目录包含索引，然后将它传递给IndexSearcher，它使用IndexReader打开目录。然后，创建一个期限查询，使搜索usingIndexSearcher通过将查询到的搜索。 IndexSearcher返回TopDocs对象包含搜索信息连同它是搜索操作的结果的文档的文档ID(多个)。

现在，我们将展示一个循序渐进的过程，以获得在索引过程的理解，使用一个基本的例子。

## 创建QueryParser

- QueryParser类解析用户输入，并输入到 Lucene 理解的格式的查询。
- 创建QueryParser的对象。
- 初始化一个在此查询运行有标准的分析版本信息和索引的名字创建QueryParser对象。

```
QueryParser queryParser;

public Searcher(String indexDirectoryPath) throws IOException{

   queryParser = new QueryParser(Version.LUCENE_36,
      LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
}
```

## 创建IndexSearcher

- IndexSearcher类作为它在索引过程中创建搜索索引的核心组成部分。
- 创建IndexSearcher对象。
- 创建其应指向位置，其中索引是存储一个 lucene 的目录。
- 初始化索引目录中创建 IndexSearcher 的对象

```
IndexSearcher indexSearcher;

public Searcher(String indexDirectoryPath) throws IOException{
   Directory indexDirectory =
      FSDirectory.open(new File(indexDirectoryPath));
   indexSearcher = new IndexSearcher(indexDirectory);
}
```

## 搜索

- 要开始搜索，通过 QueryParser 解析搜索表达式创建一个查询对象。
- 通过调用IndexSearcher.search()方法搜索。

```
Query query;

public TopDocs search( String searchQuery) throws IOException, Pars
   query = queryParser.parse(searchQuery);
   return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
}
```

## 获取文件

```
public Document getDocument(ScoreDoc scoreDoc)
   throws CorruptIndexException, IOException{
   return indexSearcher.doc(scoreDoc.doc);
}
```

## 关闭IndexSearcher

```
public void close() throws IOException{
   indexSearcher.close();
}
```

## 应用程序示例

让我们创建一个测试Lucene的应用程序来测试搜索过程。

| 步骤 | 描述 |
|---|---|
| 1 | 创建下名称为LuceneFirstApplication的一个项目作为解释Lucene的应用在包packagecom.yiibai.lucene下，在第一个应用程序的篇章。也可以使用Lucene创建的项目理解搜索过程。 |
| 2 | 创建LuceneConstants.java，TextFileFilter.java和Searcher.java 用于Lucene解释- 在第一应用章节。其它文件保持不变。 |
| 3 | 创建LuceneTester.java如下所述。 |
| 4 | 清理和构建应用程序，以确保业务逻辑按要求工作。 |

*LuceneConstants.java*

这个类是用来提供可应用于示例应用程序中使用的各种常量。

```
package com.yiibai.lucene;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

*TextFileFilter.java*

此类用于 .txt文件过滤器。

```
package com.yiibai.lucene;

import java.io.File;
import java.io.FileFilter;

public class TextFileFilter implements FileFilter {

   @Override
   public boolean accept(File pathname) {
      return pathname.getName().toLowerCase().endsWith(".txt");
   }
}
```

*Searcher.java*

这个类用来读取就使用Lucene库的原始数据，并搜索数据的索引。

```
package com.yiibai.lucene;

import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Searcher {

   IndexSearcher indexSearcher;
   QueryParser queryParser;
   Query query;

   public Searcher(String indexDirectoryPath) throws IOException{
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));
      indexSearcher = new IndexSearcher(indexDirectory);
      queryParser = new QueryParser(Version.LUCENE_36,
         LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
   }

   public TopDocs search( String searchQuery)
      throws IOException, ParseException{
      query = queryParser.parse(searchQuery);
      return indexSearcher.search(query, LuceneConstants.MAX_SEARCH
   }

   public Document getDocument(ScoreDoc scoreDoc)
      throws CorruptIndexException, IOException{
     return indexSearcher.doc(scoreDoc.doc);
   }

   public void close() throws IOException{
      indexSearcher.close();
   }
}
```

*LuceneTester.java*

这个类是用来测试 Lucene 库的搜索能力。

```
package com.yiibai.lucene;

import java.io.IOException;

import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;

public class LuceneTester {

   String indexDir = "E:\Lucene\Index";
   String dataDir = "E:\Lucene\Data";
   Searcher searcher;

   public static void main(String[] args) {
      LuceneTester tester;
      try {
         tester = new LuceneTester();
         tester.search("Mohan");
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }

   private void search(String searchQuery) throws IOException, Pars
      searcher = new Searcher(indexDir);
      long startTime = System.currentTimeMillis();
      TopDocs hits = searcher.search(searchQuery);
      long endTime = System.currentTimeMillis();

      System.out.println(hits.totalHits +
         " documents found. Time :" + (endTime - startTime) +" ms")
      for(ScoreDoc scoreDoc : hits.scoreDocs) {
         Document doc = searcher.getDocument(scoreDoc);
         System.out.println("File: "+ doc.get(LuceneConstants.FILE_
      }
      searcher.close();
   }
}
```

## 数据和索引目录的创建

从record1.txt到record10.txt的文件中包含简单的名称以及学生的其他细节，并把它们放在目录E:LuceneData。这是测试数据。索引目录路径应创建为E:LuceneIndex。期间，运行 Lucene索引程序后- 索引过程中，可以看到该文件夹中创建的索引文件的列表。

## 运行程序：

一旦创建源，创造了原始数据，数据目录，索引目录和索引完成后，已经准备好这一步是编译和运行程序。要做到这一点，请LuceneTester.Java文件选项卡中使用Eclipse IDE可使用Run选项，或使用Ctrl+ F11来编译和运行应用程序LuceneTester。如果您的应用程序一切正常，这将在Eclipse IDE的控制台打印以下消息：

```
1 documents found. Time :29 ms
File: E:LuceneData
ecord4.txt
```

# Lucene查询编程 - Lucene教程

正如我们已经看到在前面的章节中的Lucene- 搜索操作，Lucene使用IndexSearcher进行搜索，并使用由QueryParser输入创建的查询对象。在本章中，我们将讨论不同类型的查询对象和方法以编程方式来创建它们。创建不同类型的查询对象的给出了要进行搜索类型的控制。

考虑高级搜索的情况下，许多应用程序，用户给出了多个选项来限制搜索结果中提供。通过查询程序，我们一样可以很轻松的实现。

以下是查询类型，我们将在适当的时候讨论的列表。

| Sr. No. | 类和说明 |
|---|---|
| 1 | TermQuery 此类充当创造/在索引处理更新指标的核心组成部分。 |
| 2 | TermRangeQuery TermRangeQuery是在使用的范围内的文本的词条都被搜索。 |
| 3 | PrefixQuery PrefixQuery用于匹配其索引开始以指定的字符串的文档。 |
| 4 | BooleanQuery BooleanQuery用于搜索的是使用AND，OR或NOT运算符多个查询结果的文件。 |
| 5 | PhraseQuery 词组查询用于搜索包含词条的特定序列的文档。 |
| 6 | WildCardQuery WildcardQuery用于搜索使用类似 '*' 的字符序列的通配符的文件，"?" 匹配单个字符。 |
| 7 | FuzzyQuery FuzzyQuery用于搜索使用模糊实现，它是一种基于编辑距离算法的近似搜索文件。 |
| 8 | MatchAllDocsQuery MatchAllDocsQuery作为顾名思义匹配的所有文件。 |

# Lucene分析 - Lucene教程

正如我们已经看到在前一章的Lucene索引过程，Lucene使用IndexWriterwhich分析用分析仪文件，然后根据需要创建/打开/编辑索引。在本章中，我们将讨论不同类型的分析对象，哪些是在分析过程中使用的相关对象。了解分析过程中，分析仪如何工作，会给Lucene索引文件很大的启示。

以下是我们将在适当的时候讨论对象的列表。

| Sr. No. | 类和说明 |
|---|---|
| 1 | Token 令牌表示(起始偏移量，结束偏移，令牌类型和位置增量位置，)在像它的元数据相关的详细信息的文档中的文本或字。 |
| 2 | TokenStream TokenStream是分析过程中的一个输出，它包括串联的令牌。它是一个抽象类。 |
| 3 | Analyzer 这是对每个类型分析器的抽象基类。 |
| 4 | WhitespaceAnalyzer 该分析仪analyzer分割的基础的空白文档中的文本。 |
| 5 | SimpleAnalyzer 此分析器分割在基于非字母字符的文档的文本，然后小写它们。 |
| 6 | StopAnalyzer 该分析仪的工作原理类似于SimpleAnalyzer并删除常用词像 'a','an','the'等等。 |
| 7 | StandardAnalyzer 这是最复杂的分析，并能处理姓名，电子邮件地址等，它小写每个标记，并删除常用词和标点符号(如有)。 |

# Lucene排序 - Lucene教程

在本章中，我们将研究的排序顺序默认情况下，在给出的Lucene搜索结果或可根据需要进行操作。

## 按相关性排序

这是使用Lucene的默认排序方式。 Lucene通过在顶部的最相关的命中提供的结果。

```
private void sortUsingRelevance(String searchQuery)
   throws IOException, ParseException{
   searcher = new Searcher(indexDir);
   long startTime = System.currentTimeMillis();
   //create a term to search file name
   Term term = new Term(LuceneConstants.FILE_NAME, searchQuery);
   //create the term query object
   Query query = new FuzzyQuery(term);
   searcher.setDefaultFieldSortScoring(true, false);
   //do the search
   TopDocs hits = searcher.search(query,Sort.RELEVANCE);
   long endTime = System.currentTimeMillis();

   System.out.println(hits.totalHits +
      " documents found. Time :" + (endTime - startTime) + "ms");
   for(ScoreDoc scoreDoc : hits.scoreDocs) {
      Document doc = searcher.getDocument(scoreDoc);
      System.out.print("Score: "+ scoreDoc.score + " ");
      System.out.println("File: "+ doc.get(LuceneConstants.FILE_PAT
   }
   searcher.close();
}
```

## 选择IndexOrder排序方式

这被分拣使用lucene模式，其中索引第一文档示出第一搜索结果中。

```
private void sortUsingIndex(String searchQuery)
   throws IOException, ParseException{
   searcher = new Searcher(indexDir);
   long startTime = System.currentTimeMillis();
   //create a term to search file name
   Term term = new Term(LuceneConstants.FILE_NAME, searchQuery);
   //create the term query object
   Query query = new FuzzyQuery(term);
   searcher.setDefaultFieldSortScoring(true, false);
   //do the search
   TopDocs hits = searcher.search(query,Sort.INDEXORDER);
   long endTime = System.currentTimeMillis();

   System.out.println(hits.totalHits +
      " documents found. Time :" + (endTime - startTime) + "ms");
   for(ScoreDoc scoreDoc : hits.scoreDocs) {
      Document doc = searcher.getDocument(scoreDoc);
      System.out.print("Score: "+ scoreDoc.score + " ");
      System.out.println("File: "+ doc.get(LuceneConstants.FILE_PAT
   }
   searcher.close();
}
```

# 应用程序示例...

让我们创建一个测试Lucene的应用程序来测试分选过程。

| 步骤 | 描述 |
|---|---|
| 1 | 创建下名称为LuceneFirstApplication的一个项目作为解释Lucene的应用在包packagecom.yiibai.lucene下，在第一个应用程序的篇章。也可以使用Lucene创建的项目理解搜索过程。 |
| 2 | 创建LuceneConstants.java和Searcher.java作为Lucene的解释- 第一应用程序一章。保持其它文件不变。 |
| 3 | 创建LuceneTester.java如下所述。 |
| 4 | 清理和构建应用程序，以确保业务逻辑按要求工作。 |

*LuceneConstants.java*

这个类是用来提供可应用于示例应用程序中使用的各种常量。

```
package com.yiibai.lucene;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

*Searcher.java*

这个类用来读取就使用Lucene库的原始数据，并搜索数据的索引。

```
package com.yiibai.lucene;

import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.Sort;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Searcher {

IndexSearcher indexSearcher;
   QueryParser queryParser;
   Query query;

   public Searcher(String indexDirectoryPath) throws IOException{
      Directory indexDirectory
         = FSDirectory.open(new File(indexDirectoryPath));
      indexSearcher = new IndexSearcher(indexDirectory);
      queryParser = new QueryParser(Version.LUCENE_36,
         LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
   }

   public TopDocs search( String searchQuery)
      throws IOException, ParseException{
      query = queryParser.parse(searchQuery);
      return indexSearcher.search(query, LuceneConstants.MAX_SEARCH
```

```
    }

    public TopDocs search(Query query)
       throws IOException, ParseException{
       return indexSearcher.search(query, LuceneConstants.MAX_SEARCH
    }

    public TopDocs search(Query query,Sort sort)
       throws IOException, ParseException{
       return indexSearcher.search(query,
          LuceneConstants.MAX_SEARCH,sort);
    }

    public void setDefaultFieldSortScoring(boolean doTrackScores,
       boolean doMaxScores){
       indexSearcher.setDefaultFieldSortScoring(
          doTrackScores,doMaxScores);
    }

    public Document getDocument(ScoreDoc scoreDoc)
       throws CorruptIndexException, IOException{
       return indexSearcher.doc(scoreDoc.doc);
    }

    public void close() throws IOException{
       indexSearcher.close();
    }
}
```

*LuceneTester.java*

这个类是用来测试Lucene库的搜索能力。

```
package com.yiibai.lucene;

import java.io.IOException;

import org.apache.lucene.document.Document;
import org.apache.lucene.index.Term;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.FuzzyQuery;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.Sort;
import org.apache.lucene.search.TopDocs;

public class LuceneTester {

   String indexDir = "E:\Lucene\Index";
   String dataDir = "E:\Lucene\Data";
   Indexer indexer;
```

```
    Searcher searcher;

    public static void main(String[] args) {
       LuceneTester tester;
       try {
           tester = new LuceneTester();
           tester.sortUsingRelevance("cord3.txt");
           tester.sortUsingIndex("cord3.txt");
       } catch (IOException e) {
           e.printStackTrace();
       } catch (ParseException e) {
           e.printStackTrace();
       }
    }

    private void sortUsingRelevance(String searchQuery)
       throws IOException, ParseException{
       searcher = new Searcher(indexDir);
       long startTime = System.currentTimeMillis();
       //create a term to search file name
       Term term = new Term(LuceneConstants.FILE_NAME, searchQuery);
       //create the term query object
       Query query = new FuzzyQuery(term);
       searcher.setDefaultFieldSortScoring(true, false);
       //do the search
       TopDocs hits = searcher.search(query,Sort.RELEVANCE);
       long endTime = System.currentTimeMillis();

       System.out.println(hits.totalHits +
          " documents found. Time :" + (endTime - startTime) + "ms")
       for(ScoreDoc scoreDoc : hits.scoreDocs) {
          Document doc = searcher.getDocument(scoreDoc);
          System.out.print("Score: "+ scoreDoc.score + " ");
          System.out.println("File: "+ doc.get(LuceneConstants.FILE_
       }
       searcher.close();
    }

    private void sortUsingIndex(String searchQuery)
       throws IOException, ParseException{
       searcher = new Searcher(indexDir);
       long startTime = System.currentTimeMillis();
       //create a term to search file name
       Term term = new Term(LuceneConstants.FILE_NAME, searchQuery);
       //create the term query object
       Query query = new FuzzyQuery(term);
       searcher.setDefaultFieldSortScoring(true, false);
       //do the search
       TopDocs hits = searcher.search(query,Sort.INDEXORDER);
       long endTime = System.currentTimeMillis();

       System.out.println(hits.totalHits +
       " documents found. Time :" + (endTime - startTime) + "ms");
```

```
        for(ScoreDoc scoreDoc : hits.scoreDocs) {
           Document doc = searcher.getDocument(scoreDoc);
           System.out.print("Score: "+ scoreDoc.score + " ");
           System.out.println("File: "+ doc.get(LuceneConstants.FILE_
        }
        searcher.close();
     }
  }
```

## 数据和索引目录的创建

从record1.txt到record10.txt的文件中包含简单的名称以及学生的其他细节，并把它们放在目录E:LuceneData。这是测试数据。索引目录路径应创建为E:LuceneIndex。期间，运行 Lucene索引程序后- 索引过程中，可以看到该文件夹中创建的索引文件的列表。

## 运行程序：

一旦创建源，创造了原始数据，数据目录，索引目录和索引完成后，已经准备好这一步是编译和运行程序。要做到这一点，请LuceneTester.Java文件选项卡中使用Eclipse IDE可使用Run选项，或使用Ctrl+ F11来编译和运行应用程序LuceneTester。如果您的应用程序一切正常，这将在Eclipse IDE的控制台打印以下消息：

```
10 documents found. Time :31ms
Score: 1.3179655 File: E:LuceneData
ecord3.txt
Score: 0.790779 File: E:LuceneData
ecord1.txt
Score: 0.790779 File: E:LuceneData
ecord2.txt
Score: 0.790779 File: E:LuceneData
ecord4.txt
Score: 0.790779 File: E:LuceneData
ecord5.txt
Score: 0.790779 File: E:LuceneData
ecord6.txt
Score: 0.790779 File: E:LuceneData
ecord7.txt
Score: 0.790779 File: E:LuceneData
ecord8.txt
Score: 0.790779 File: E:LuceneData
ecord9.txt
Score: 0.2635932 File: E:LuceneData
ecord10.txt
10 documents found. Time :0ms
Score: 0.790779 File: E:LuceneData
ecord1.txt
Score: 0.2635932 File: E:LuceneData
ecord10.txt
Score: 0.790779 File: E:LuceneData
ecord2.txt
Score: 1.3179655 File: E:LuceneData
ecord3.txt
Score: 0.790779 File: E:LuceneData
ecord4.txt
Score: 0.790779 File: E:LuceneData
ecord5.txt
Score: 0.790779 File: E:LuceneData
ecord6.txt
Score: 0.790779 File: E:LuceneData
ecord7.txt
Score: 0.790779 File: E:LuceneData
ecord8.txt
Score: 0.790779 File: E:LuceneData
ecord9.txt
```

# Maven教程

Apache Maven是一个软件项目管理和综合工具。基于项目对象模型（POM）的概念，Maven可以从一个中心资料片管理项目构建，报告和文件。

本教程将介绍如何使用Maven在Java开发，或任何其他的编程语言的任何项目。

# Maven是什么？

Maven是一个项目管理和综合工具。Maven提供了开发人员构建一个完整的生命周期框架。开发团队可以自动完成项目的基础工具建设，Maven使用标准的目录结构和默认构建生命周期。

在多个开发团队环境时，Maven可以设置按标准在非常短的时间里完成配置工作。由于大部分项目的设置都很简单，并且可重复使用，Maven让开发人员的工作更轻松，同时创建报表，检查，构建和测试自动化设置。

Maven提供了开发人员的方式来管理：

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- mailing list

概括地说，Maven简化和标准化项目建设过程。处理编译，分配，文档，团队协作和其他任务的无缝连接。 Maven增加可重用性并负责建立相关的任务。

# Maven历史

Maven最初设计，是以简化Jakarta Turbine项目的建设。在几个项目，每个项目包含了不同的Ant构建文件。 JAR检查到CVS。

Apache组织开发Maven可以建立多个项目，发布项目信息，项目部署，在几个项目中JAR文件提供团队合作和帮助。

## Maven目标

Maven主要目标是提供给开发人员：

- 项目是可重复使用，易维护，更容易理解的一个综合模型。
- 插件或交互的工具，这种声明性的模式。

Maven项目的结构和内容在一个XML文件中声明，pom.xml 项目对象模型（POM），这是整个Maven系统的基本单元。有关详细信息，请参阅Maven POM的部分。

Apache Maven 是一种创新的软件项目管理工具，提供了一个项目对象模型（POM）文件的新概念来管理项目的构建，相关性和文档。最强大的功能就是能够自动下载项目依赖库。

在本教程中，它提供了如何使用 Apache Maven 3.x 的许多实例和解释。

## Maven安装和配置

在 Windows 和 Ubuntu 的安装指南。

- 在Windows上安装Maven 有关如何在 Windows 上安装 Maven 的文章。
- 启用Maven的代理访问 要使用代理服务器来连接互联网，必须在 Maven 配置代理设置。

## Maven资源库

Maven 位置，中央和远程存储库配置和解释，有些术语可能需要在 Maven 使用前理解。

- Maven本地资源库 Maven 的本地资源库是用来存储项目的依赖库，默认的文件夹是 ".m2" 目录，可能需要将其更改为另一个文件夹。
- Maven中央存储库 Maven 中央存储库是 Maven 用来下载所有项目的依赖库的默认位置。
- 如何从Maven远程存储库下载？，如何添加远程库？ 并非所有的库存储在Maven的中央存储库，很多时候需要添加一些远程仓库来从其他位置，而不是默认的中央存储库下载库。
- Maven依赖机制 这里的文章是关于传统方式和Maven方式的依赖库的不同，并说明 Maven 会从那里搜索这些库。
- 定制库到Maven本地资源库 很多库仍然不支持 Maven 的 pom.xml 的概念，这里有一个指南来说明如何包括"非Maven支持"库到 Maven 本地资源库中。

## 基于Maven项目和Eclipse IDE

实例是使用Maven创建Java项目和Web应用程序，以及演示如何将其导入到Eclipse IDE中。

- 使用Maven创建Java项目 使用 Maven 来创建一个 Java 项目。
- 转换基于Maven的Java项目支持Eclipse IDE 指导转换基于Maven的Java项目来支持在 Eclipse IDE 中。
- 使用Maven创建Web应用程序项目 使用Maven来创建Web应用程序项目。
- 转换基于Maven的Web应用程序支持Eclipse IDE Maven转换基于Web应用程序来支持Eclipse IDE中指南。
- 使用Maven模板创建项目 另外，您也可以从 Maven 的模板来创建标准项目。

## Maven基本操作

一些基本的操作，编译，构建，单元测试，安装，网站生成和基于Maven部署项目。

- 使用Maven构建项目 “mvn package” 来构建项目
- 使用Maven清理项目 “mvn clean” 来清理项目
- 使用Maven运行单元测试 “mvn test” 来执行单元测试
- 将项目安装到Maven本地资源库 “mvn install” 打包和部署项目到本地资源库
- 生成基于Maven的项目文档站点 “mvn site” 来为您的项目生成信息文档站点
- 使用“mvn site-deploy”部署站点（WebDAV例子） “mvn site-deploy” 通过WebDAV部署自动生成的文档站点到服务器
- 部署基于Maven的war文件到Tomcat “mvn tomcat:deploy” 以 WAR 文件部署到Tomcat

## Maven参考

- Apache Maven 官方教程
- Apache Maven (Wiki)

# Maven安装配置 - Maven教程

想要安装 Apache Maven 在Windows 系统上, 只需要下载 Maven 的 zip 文件，并将其解压到你想安装的目录，并配置 Windows 环境变量。

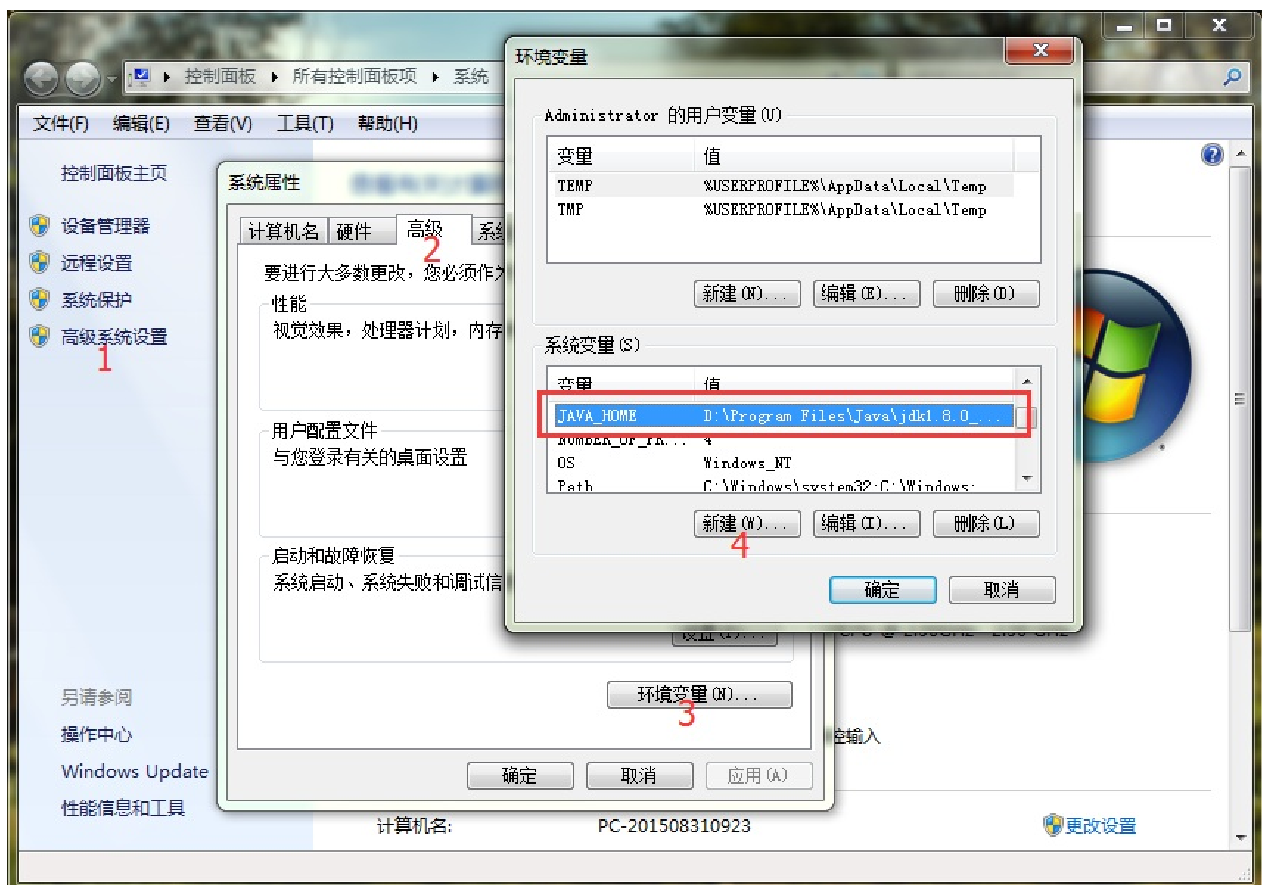
所需工具 ：

1. JDK 1.8
2. Maven 3.3.3
3. Windows 7

注 Maven 3.2 要求 JDK 1.6 或以上版本, 而Maven 3.0/3.1 需要 JDK 1.5 或以上

## 1. JDK 和 JAVA_HOME

确保已安装JDK，并 “JAVA_HOME” 变量已加入作为 Windows 环境变量。




操作要以按上面数字顺序，在这个教程中，安装的 JDK 是 JDK1.8，为了方便学习，建议你也安装使用 JDK1.8。

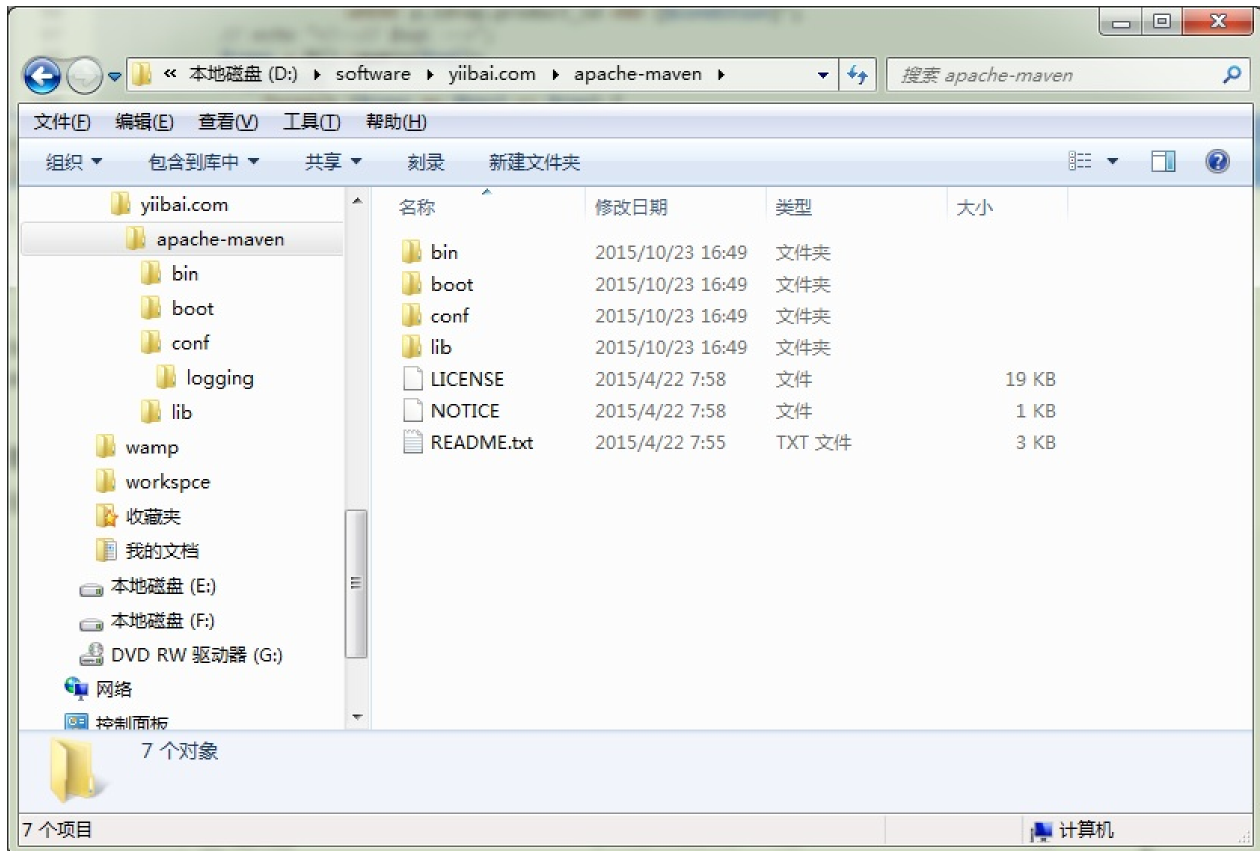
## 2. 下载Apache Maven

访问 Maven官方网站，打开后找到下载链接，如下：



下载 Maven 的 zip 文件，例如：apache-maven-3.3.3-bin.zip，将它解压到你要安装 Maven 的文件夹。

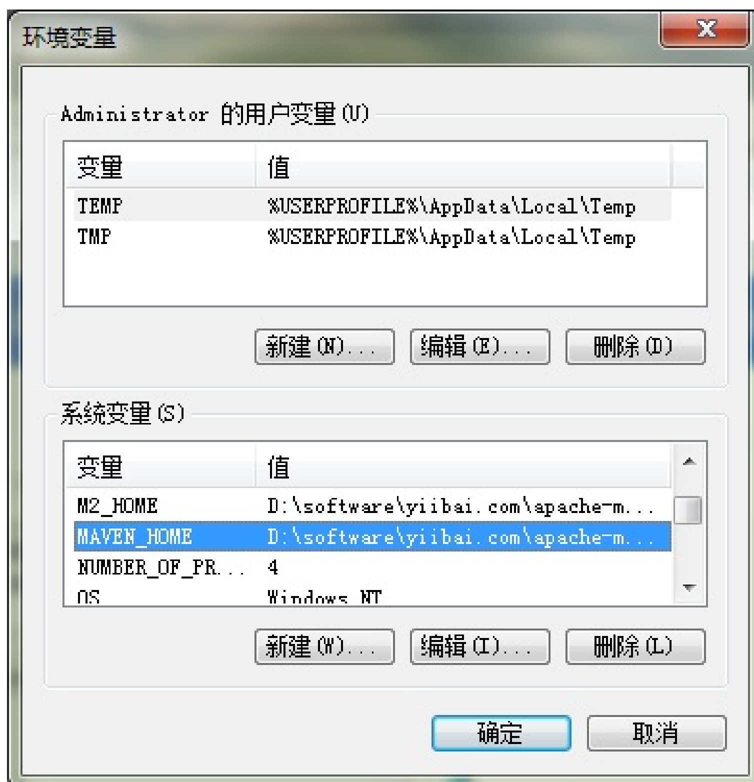
假设你解压缩到这个文件夹 – D:\software\yiibai.com\apache-maven




注意：在这一步，只是文件夹和文件，安装不是必需的

## 3. 添加 M2_HOME 和 MAVEN_HOME

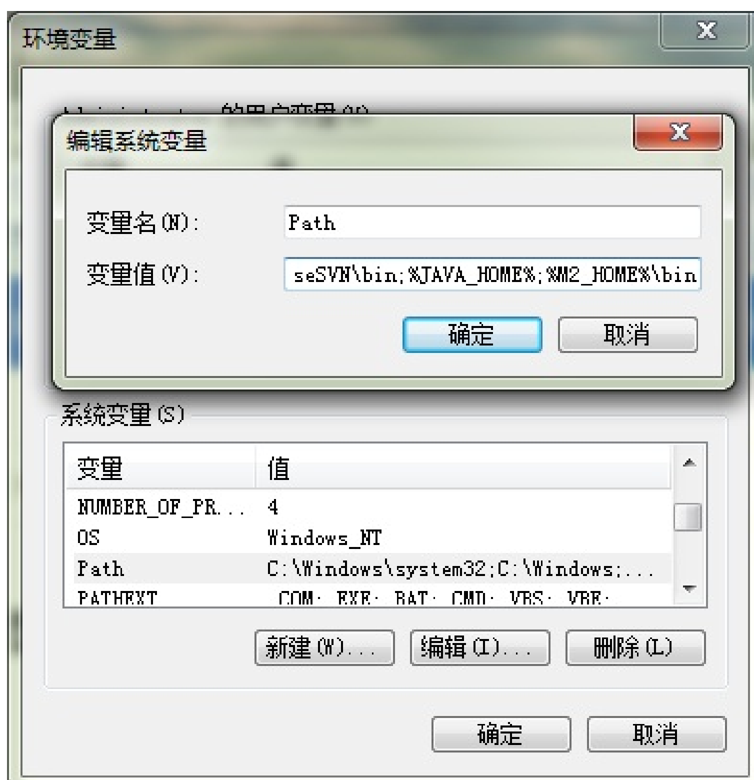
添加 M2_HOME 和 MAVEN_HOME 环境变量在Windows，并将其指向你的 Maven 文件夹。




M2_HOME 或 MAVEN_HOME Maven 说只是添加 M2_HOME , 但一些项目仍引用 Maven 的文件夹 MAVEN_HOME, 因此，为了安全也把它添加进去。

# 4. 添加到环境变量 - PATH

更新 PATH 变量，添加 Maven bin 文件夹到 PATH 的最后 – %M2_HOME%\bin, 这样就可以到处运行 Maven 命令了。



# 5. 验证

完成，以验证它，执行 mvn –version 在命令提示符下，如下图输出结果：

```
C:\Users\Administrator>mvn -version
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-
7+08:00)
Maven home: D:\software\yiibai.com\apache-maven
Java version: 1.8.0_40, vendor: Oracle Corporation
Java home: D:\Program Files\Java\jdk1.8.0_40
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
```

如果你看到类似消息，表示 Apache Maven 在 Windows 上已安装成功。

# Maven启用代理访问 - Maven教程

如果你的公司正在建立一个防火墙，并使用HTTP代理服务器来阻止用户直接连接到互联网。如果您使用代理，Maven将无法下载任何依赖。

为了使它工作，你必须声明在 Maven 的配置文件中设置代理服务器：settings.xml.

## 1. Maven配置文件

找到文件 {M2_HOME}/conf/settings.xml, 并把你的代理服务器信息配置写入。注：{M2_HOME} => D:\software\yiibai.com\apache-maven

{M2_HOME}/conf/settings.xml

```
<!-- proxies
   | This is a list of proxies which can be used on this machine to
   | Unless otherwise specified (by system property or command-line
   | specification in this list marked as active will be used.
   |-->
  <proxies>
    <!-- proxy
     | Specification for one proxy, to be used in connecting to the
     |
    <proxy>
      <id>optional</id>
      <active>true</active>
      <protocol>http</protocol>
      <username>proxyuser</</username>
      <password>proxypass</password>
      <host>proxy.host.net</host>
      <port>80</port>
      <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
    </proxy>
    -->
  </proxies>
```

取消注释代理选项，填写您的代理服务器的详细信息。

```
<!-- proxies
   | This is a list of proxies which can be used on this machine to
   | Unless otherwise specified (by system property or command-line
   | specification in this list marked as active will be used.
   |-->
  <proxies>
      <proxy>
      <id>optional</id>
      <active>true</active>
      <protocol>http</protocol>
      <username>yiibai</username>
      <password>password</password>
      <host>proxy.yiibai.com</host>
      <port>8888</port>
      <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
    </proxy>
  </proxies>
```

## 2. 保存文件

完成后，Apache Maven 应该是能够通过代理服务器立即连接到Internet。

注意：重新启动不是必需的。Maven 只是一个命令，当你调用它，它会再次读取该文件。

# Maven本地资源库 - Maven教程

Maven的本地资源库是用来存储所有项目的依赖关系(插件jar和其他文件，这些文件被Maven下载)到本地文件夹。很简单，当你建立一个Maven项目，所有相关文件将被存储在你的Maven本地仓库。

默认情况下，Maven的本地资源库默认为 .m2 目录文件夹：

1. Unix/Mac OS X – ~/.m2
2. Windows – C:\Documents and Settings{your-username}.m2

## 1. 更新Maven的本地库

通常情况下，可改变默认的 .m2 目录下的默认本地存储库文件夹到其他更有意义的名称，例如， maven-repo

找到 {M2_HOME}\conf\setting.xml, 更新 localRepository 到其它名称。

{M2_HOME}\conf\setting.xml

```
<settings><!-- localRepository
    | The path to the local repository maven will use to store artif
    |
    | Default: ~/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  --><localRepository>D:\software\yiibai.com\apache-maven\repositor
```

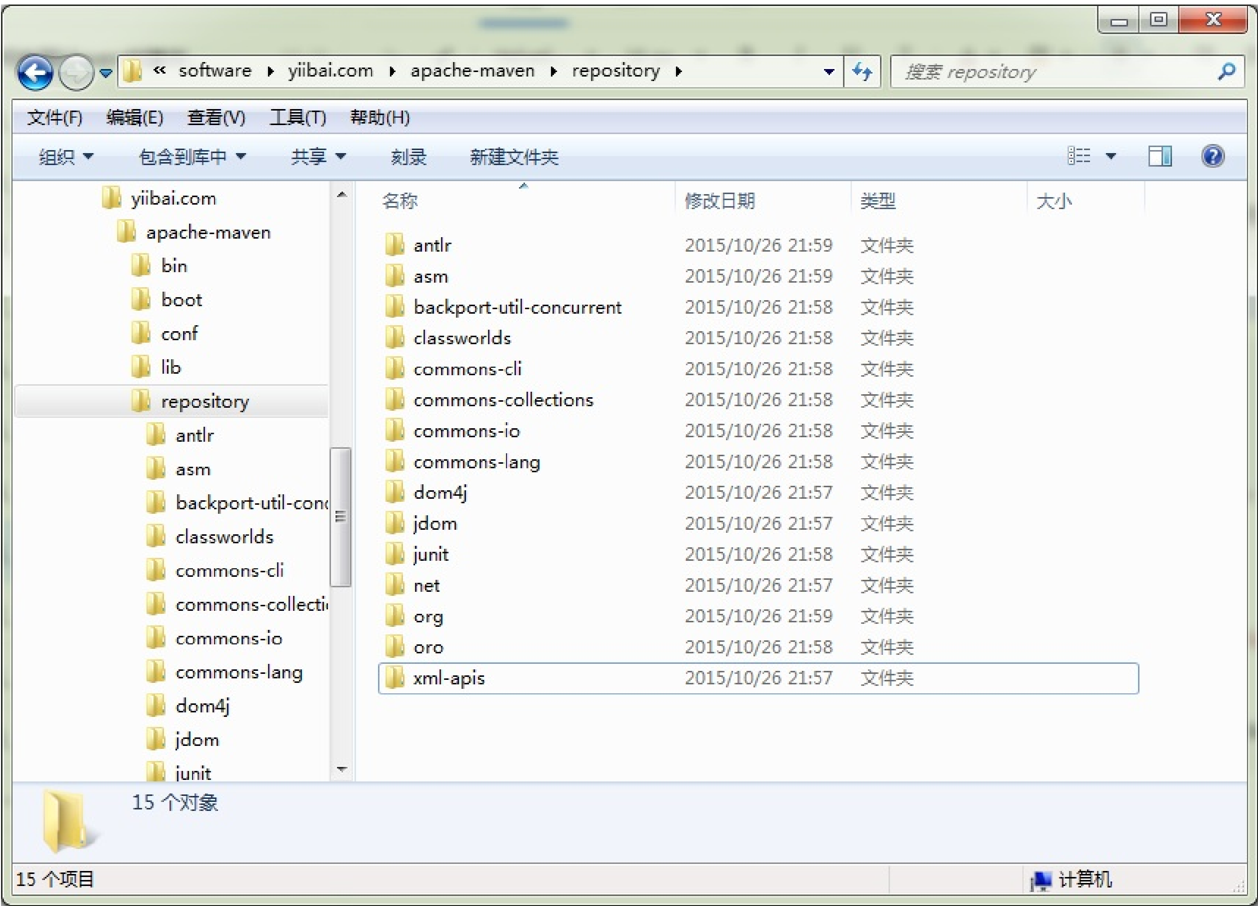
## 2. 保存文件

执行之后，新的 Maven 本地存储库现在改为 D:\software\yiibai.com\apache-maven\repository.

执行命令：

```
C:\worksp> mvn archetype:generate -DgroupId=com.yiibai -DartifactId
```

详见如下图：

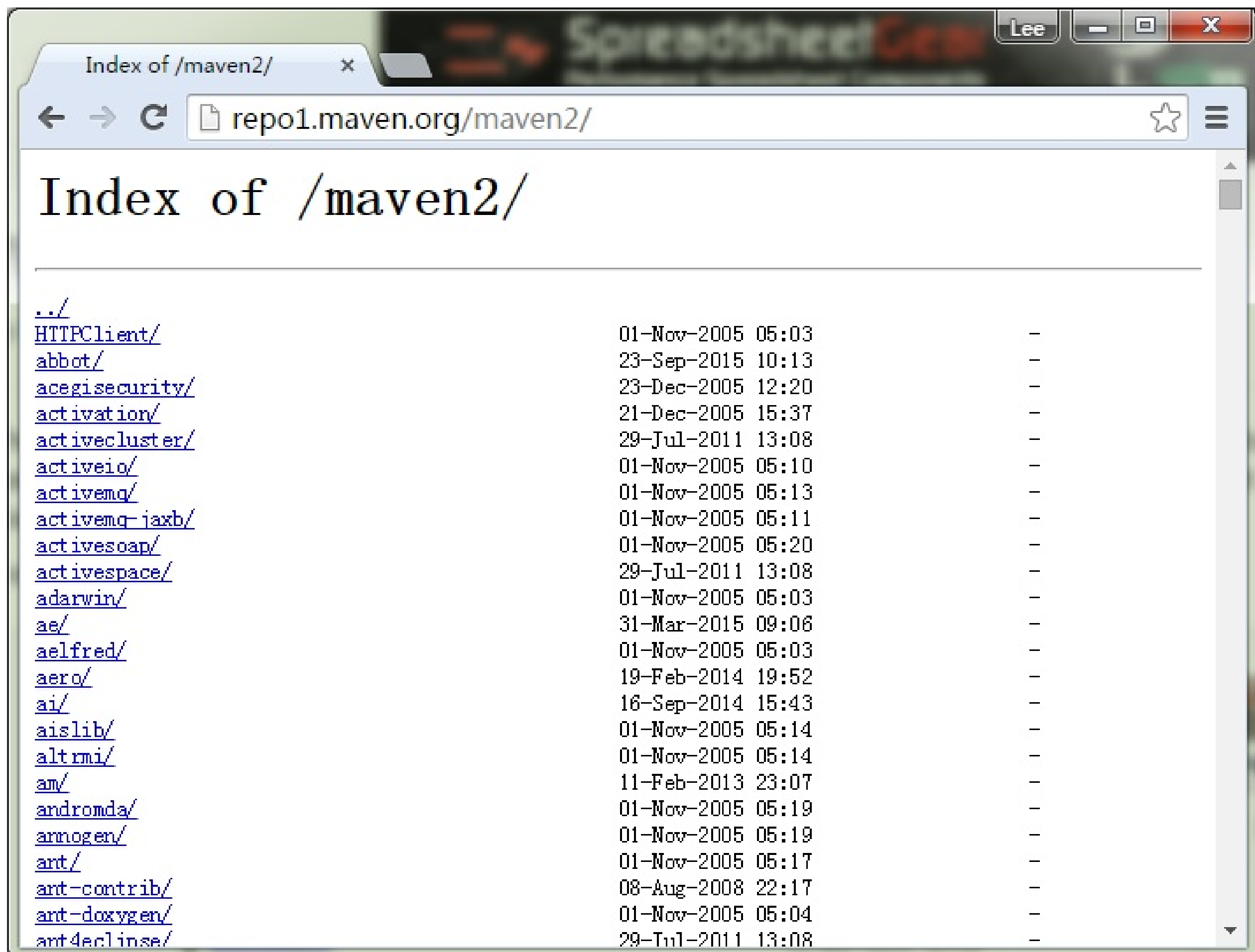


« software ▸ yiibai.com ▸ apache-maven ▸ repository ▸
搜索 repository
文件(F) 编辑(E) 查看(V) 工具(T) 帮助(H)
组织 包含到库中 共享 刻录 新建文件夹
yiibai.com
apache-maven
bin
boot
conf
lib
repository
antlr
asm
backport-util-conc
classworlds
commons-cli
commons-collecti
commons-io
commons-lang
dom4j
jdom
junit
名称 修改日期 类型 大小
antlr 2015/10/26 21:59 文件夹
asm 2015/10/26 21:59 文件夹
backport-util-concurrent 2015/10/26 21:58 文件夹
classworlds 2015/10/26 21:58 文件夹
commons-cli 2015/10/26 21:58 文件夹
commons-collections 2015/10/26 21:58 文件夹
commons-io 2015/10/26 21:58 文件夹
commons-lang 2015/10/26 21:58 文件夹
dom4j 2015/10/26 21:57 文件夹
jdom 2015/10/26 21:57 文件夹
junit 2015/10/26 21:58 文件夹
net 2015/10/26 21:57 文件夹
org 2015/10/26 21:59 文件夹
oro 2015/10/26 21:58 文件夹
xml-apis 2015/10/26 21:57 文件夹
15 个对象
15 个项目
计算机

# Maven中央存储库 - Maven教程

当你建立一个 Maven 的项目，Maven 会检查你的 pom.xml 文件，以确定哪些依赖下载。首先，Maven 将从本地资源库获得 Maven 的本地资源库依赖资源，如果没有找到，然后把它会从默认的 Maven 中央存储库 – http://repo1.maven.org/maven2/ 查找下载。

Maven 的中央资源库网站是这样的：




Maven中心储存库网站已经改版本，目录浏览可能不再使用。这将直接被重定向到 http://search.maven.org/。这就好多了，现在有一个搜索功能：

The Central Repository ×

search.maven.org/#search%7Cga%7C1%7Ca

# The Central Repository

SEARCH | ADVANCED SEARCH | BROWSE

a

SEARCH

New: About Central    Advanced Search | API Guide | Help

## Search Results

< 1 2 3 ... 15 16 > displa

| GroupId | ArtifactId | Latest Version | Updated | Download |
|---|---|---|---|---|
| com.carrotgarden.a | carrot-a | 1.5.31 | 19-Nov-2014 | pom |
| com.ajipi.a-sysmon | a-sysmon-parent | 1.0-pre28 all (27) | 01-Sep-2015 | pom |
| com.ajipi.a-foundation | a-foundation-parent | 1.0-pre16 all (3) | 01-Aug-2015 | pom |
| org.jibx.schema.org.opentravel._2011A | opentravel-2011A-reactor | 1.1.4 all (6) | 31-Dec-2014 | pom |
| org.jibx.schema.org.opentravel._2012A | opentravel-2012A-reactor | 1.1.4 all (6) | 31-Dec-2014 | pom |
| org.jibx.schema.org.opentravel._2013A | opentravel-2013A-reactor | 1.1.4 all (2) | 31-Dec-2014 | pom |
| org.jibx.schema.org.opentravel._2014A | opentravel-2014A-reactor | 1.1.4 all (2) | 31-Dec-2014 | pom |
| net.sf.a2j | a2j | beta2 | 21-Dec-2005 | pom jar |
| xfire | xfire-jaxb | 1.0 | 07-Dec-2005 | pom M6a.jar M6a.pom |
| org.ow2.joram | a3 | 5.10.0 all (6) | 28-May-2015 | pom |
| org.objectweb.joram | a3 | 5.7.0 all (6) | 14-Sep-2011 | pom |
| quilt | quilt-0.6-a | 5 | 09-Nov-2005 | pom jar |

PS：目录浏览功能被禁用，但是，当你建立 Maven 的项目，它仍然会从 “http://repo1.maven.org/maven/” 得到依赖，您可以从 Maven 验证输出。

# 如何从Maven远程存储库下载？ - Maven教程

根据 Apache Maven 的说明:

> Downloading in Maven is triggered by a project declaring a dependency that is not present in the local repository (or for a SNAPSHOT, when the remote repository contains one that is newer). By default, Maven will download from the central repository.

在Maven中，当你声明的库不存在于本地存储库中，也没有不存在于Maven中心储存库，该过程将停止并将错误消息输出到 Maven 控制台。

## 1. 示例

org.jvnet.localizer 只适用于 Java.net资源库

pom.xml

```
<dependency>
        <groupId>org.jvnet.localizer</groupId>
        <artifactId>localizer</artifactId>
        <version>1.8</version>
</dependency>
```

当你建立这个 Maven 项目，它将依赖找不到失败并输出错误消息。

## 2. 声明Java.net储存库

告诉 Maven 来获得 Java.net 的依赖，你需要声明远程仓库在 pom.xml 文件这样：

pom.xml

```
 <repositories>
    <repository>
        <id>java.net</id>
        <url>https://maven.java.net/content/repositories/public/</u
    </repository>
    </repositories>
```

现在，Maven的依赖库查询顺序更改为：

1. 在 Maven 本地资源库中搜索，如果没有找到，进入第 2 步，否则退出。
2. 在 Maven 中央存储库搜索，如果没有找到，进入第 3 步，否则退出。

3. 在java.net Maven的远程存储库搜索，如果没有找到，提示错误信息，否则退出。

# Maven添加远程仓库 - Maven教程

默认情况下，Maven从Maven中央仓库下载所有依赖关系。但是，有些库丢失在中央存储库，只有在Java.net或JBoss的储存库远程仓库中能找到。

## 1. Java.net资源库

添加Java.net远程仓库的详细信息在“pom.xml”文件。

pom.xml

```
<project ...>
<repositories>
    <repository>
      <id>java.net</id>
      <url>https://maven.java.net/content/repositories/public/</url
    </repository>
 </repositories>
</project>
```

注 旧的 “*http://download.java.net/maven/2*” 仍然可用, 但建议升级到最新储存库。

## 2. JBoss Maven存储库

1. 添加JBoss远程仓库的详细信息在 “pom.xml” 文件中。

pom.xml

```
<project ...>
    <repositories>
      <repository>
    <id>JBoss repository</id>
    <url>http://repository.jboss.org/nexus/content/groups/public/</
      </repository>
    </repositories>
</project>
```

注意：旧的 *http://repository.jboss.com/maven2/* 已过时，不再使用。

## 参考

1. http://maven2-repository.java.net/

# Maven依赖机制 - Maven教程

在 Maven 依赖机制的帮助下自动下载所有必需的依赖库，并保持版本升级。

## 案例分析

让我们看一个案例研究，以了解它是如何工作的。假设你想使用 Log4j 作为项目的日志。这里你要做什么？

## 1.在传统方式

1. 访问 http://logging.apache.org/log4j/
2. 下载 Log4 j的 jar 库
3. 复制 jar 到项目类路径
4. 手动将其包含到项目的依赖
5. 所有的管理需要一切由自己做

如果有 Log4j 版本升级，则需要重复上述步骤一次。

## 2. 在Maven的方式

1. 你需要知道 log4j 的 Maven 坐标，例如：

```
&lt;groupId&gt;log4j&lt;/groupId&gt;
&lt;artifactId&gt;log4j&lt;/artifactId&gt;
&lt;version&gt;1.2.14&lt;/version&gt;
```

2. 它会自动下载 log4j 的1.2.14 版本库。如果“version”标签被忽略，它会自动升级库时当有新的版本时。

3. 声明 Maven 的坐标转换成 pom.xml 文件。

```
&lt;dependencies&gt;
    &lt;dependency&gt;
    &lt;groupId&gt;log4j&lt;/groupId&gt;
    &lt;artifactId&gt;log4j&lt;/artifactId&gt;
    &lt;version&gt;1.2.14&lt;/version&gt;
    &lt;/dependency&gt;
&lt;/dependencies&gt;
```

4. 当 Maven 编译或构建，log4j 的 jar 会自动下载，并把它放到 Maven 本地存储库
5. 所有由 Maven 管理

## 解释说明

看看有什么不同？那么到底在Maven发生了什么？当建立一个Maven的项目，pom.xml文件将被解析，如果看到 log4j 的 Maven 坐标，然后 Maven 按此顺序搜索 log4j 库：

1. 在 Maven 的本地仓库搜索 log4j
2. 在 Maven 中央存储库搜索 log4j
3. 在 Maven 远程仓库搜索 log4j(如果在 pom.xml 中定义)

Maven 依赖库管理是一个非常好的工具，为您节省了大量的工作。

如何找到 Maven 坐标？ 访问 Maven 中心储存库，搜索下载您想要的jar。

## 参考

1. 依赖机制简介

# 定制库到Maven本地资源库 - Maven教程

这里有2个案例，需要手动发出Maven命令包括一个 jar 到 Maven 的本地资源库。

1. 要使用的 jar 不存在于 Maven 的中心储存库中。
2. 您创建了一个自定义的 jar ，而另一个 Maven 项目需要使用。

PS，还是有很多 jar 不支持 Maven 的。

## 案例学习

例如，kaptcha，它是一个流行的第三方Java库，它被用来生成 "验证码" 的图片，以阻止垃圾邮件，但它不在 Maven 的中央仓库中。

在本教程中，我们将告诉你如何安装 "kaptcha" jar 到Maven 的本地资源库。

## 1. mvn 安装

下载 "kaptcha"，将其解压缩并将 kaptcha-version.jar 复制到其他地方，比如：C盘。发出下面的命令：

```
mvn install:install-file -Dfile=c:\kaptcha-{version}.jar -DgroupId=
```

示例：

```
D:\>mvn install:install-file -Dfile=c:\kaptcha-2.3.jar -DgroupId=co
-DartifactId=kaptcha -Dversion=2.3 -Dpackaging=jar
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'install'.
[INFO] -----------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [install:install-file] (aggregator-style)
[INFO] -----------------------------------------------------------------
[INFO] [install:install-file]
[INFO] Installing c:\kaptcha-2.3.jar to
D:\maven_repo\com\google\code\kaptcha\2.3\kaptcha-2.3.jar
[INFO] -----------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -----------------------------------------------------------------
[INFO] Total time: < 1 second
[INFO] Finished at: Tue May 12 13:41:42 SGT 2014
[INFO] Final Memory: 3M/6M
[INFO] -----------------------------------------------------------------
```

现在，“kaptcha” jar被复制到 Maven 本地存储库。

## 2. pom.xml

安装完毕后，就在 pom.xml 中声明 kaptcha 的坐标。

```
<dependency>
      <groupId>com.google.code</groupId>
      <artifactId>kaptcha</artifactId>
      <version>2.3</version>
 </dependency>
```

## 3. 完成

构建它，现在 “kaptcha” jar 能够从你的 Maven 本地存储库检索了。

## 参考

1. Maven安装文档
2. Kaptcha网站

# 使用Maven创建Java项目 - Maven教程

在本教程中，我们将向你展示如何使用 Maven 来创建一个 Java 项目，导入其到 Eclipse IDE，并打包 Java 项目到一个 JAR 文件。

所需要的工具：

1. Maven 3.3.3
2. Eclipse 4.2
3. JDK 8

注意：请确保 Maven 是正确安装和配置（在Windows，*nix，Mac OSX系统中），然后再开始本教程，避免 mvn 命令未找到错误。

## 1. 从 Maven 模板创建一个项目

在终端（* UNIX或Mac）或命令提示符（Windows）中，浏览到要创建 Java 项目的文件夹。键入以下命令：

```
mvn archetype:generate -DgroupId={project-packaging} -DartifactId=
```

这告诉 Maven 来从 maven-archetype-quickstart 模板创建 Java 项目。如果忽视 archetypeArtifactId 选项，一个巨大的 Maven 模板列表将列出。

例如，这里的工作目录是：C:\worksp，执行命令过程时间可能比较久，看个人的网络状况。

```
C:\worksp>mvn archetype:generate -DgroupId=com.yiibai -DartifactId=
or -DarchetypeArtifactId=maven -archetype-quickstart -DinteractiveM
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > gene
@ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < gene
@ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ star
-
[INFO] Generating project in Batch mode
[INFO] ------------------------------------------------------------
---
[INFO] Using following parameters for creating project from Old (1
 maven-archetype-quickstart:1.0
[INFO] ------------------------------------------------------------
---
[INFO] Parameter: basedir, Value: C:\worksp
[INFO] Parameter: package, Value: com.yiibai
[INFO] Parameter: groupId, Value: com.yiibai
[INFO] Parameter: artifactId, Value: NumberGenerator
[INFO] Parameter: packageName, Value: com.yiibai
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\worksp\N
r
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 23.166 s
[INFO] Finished at: 2015-10-27T11:03:48+08:00
[INFO] Final Memory: 17M/114M
[INFO] ------------------------------------------------------------
```

在上述情况下，一个新的Java项目命名 “NumberGenerator”, 而整个项目的目录结构会自动创建。

注意 有少数用户说 mvn archetype:generate 命令未能生成项目结构。 如果您有任何类似的问题，不用担心，只需跳过此步骤，手动创建文件夹，请参阅步骤2的项目结构。

## 2.Maven目录布局

使用 mvn archetype:generate + maven-archetype-quickstart 模板, 以下项目的目录结构被创建。

```
NumberGenerator
   |-src
   |---main
   |-----java
   |-------com
   |---------yiibai
   |-----------App.java
   |---test|-----java
   |-------com
   |---------yiibai
   |-----------AppTest.java
   |-pom.xml
```

很简单的，所有的源代码放在文件夹 /src/main/java/, 所有的单元测试代码放入 /src/test/java/.

注意，请阅读 Maven标准目录布局

附加的一个标准的 pom.xml 被生成。这个POM文件类似于 Ant build.xml 文件，它描述了整个项目的信息，一切从目录结构，项目的插件，项目依赖，如何构建这个项目等，请阅读POM官方指南

pom.xml

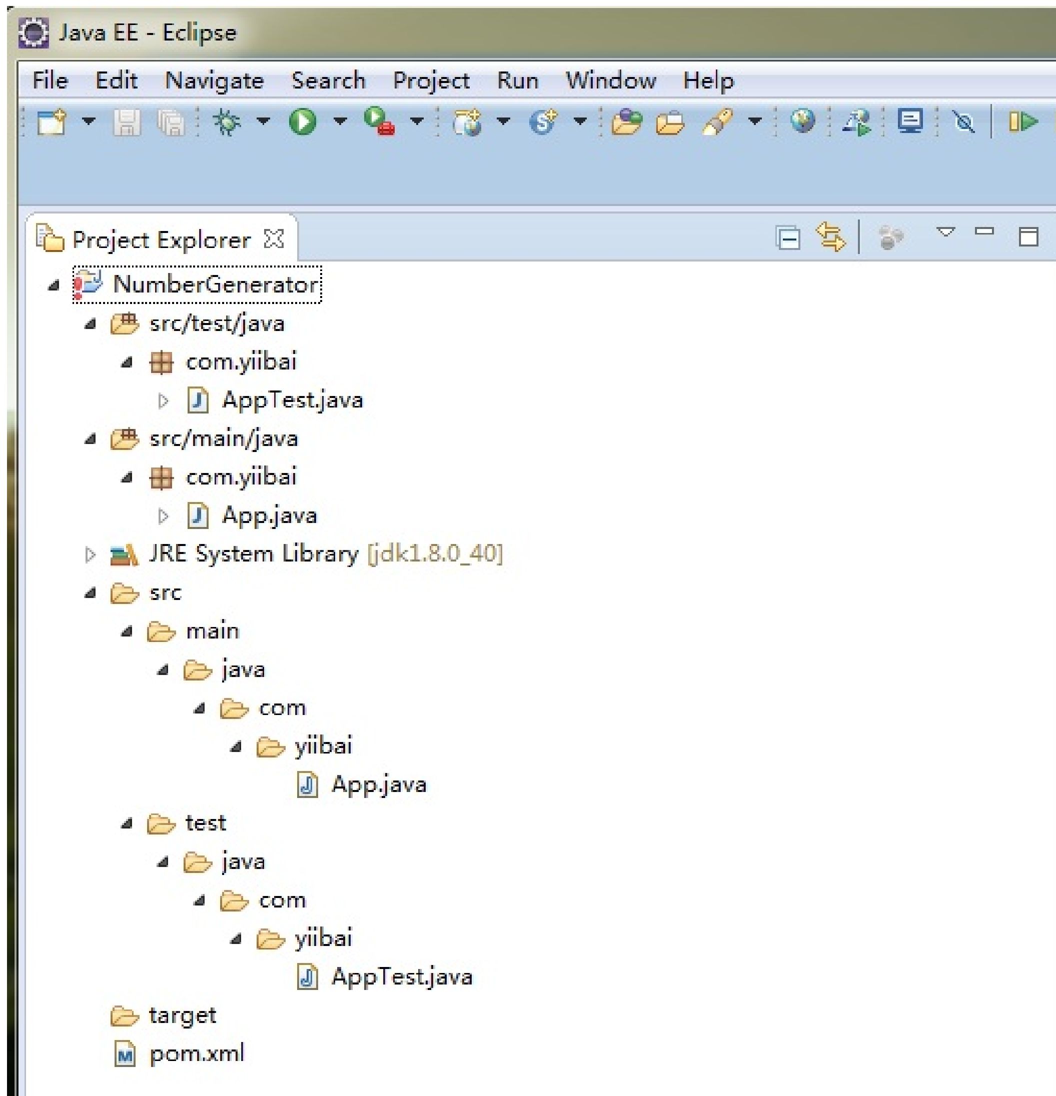
# 3. Eclipse IDE

为了使它成为一个 Eclipse 项目，在终端进入到 “NumberGenerator” 项目，键入以下命令：

```
C:\worksp\NumberGenerator>mvn eclipse:eclipse
.... ...
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launchin
ER
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "NumberGenerator" to C:\worksp\Nu
[INFO]
[INFO] -------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] -------------------------------------------------------------
[INFO] Total time: 04:47 min
[INFO] Finished at: 2015-10-27T15:24:48+08:00
[INFO] Final Memory: 15M/164M
[INFO] -------------------------------------------------------------
```

执行以上命令后，它自动下载更新相关资源和配置信息（需要等待一段时间），并产生 Eclipse IDE所要求的所有项目文件。要导入项目到Eclipse IDE中，选择 “File -> Import… -> General->Existing Projects into Workspace”

图片: 项目导入到 Eclipse IDE中。






# 4. 更新POM

默认的 pom.xml 太简单了，很多时候，你需要添加编译器插件来告诉 Maven 使用哪个 JDK 版本是用来编译项目。（默认JDK1.4，这的确太旧了点）

```
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>2.3.2</version>
          <configuration>
              <source>1.6</source>
              <target>1.6</target>
          </configuration>
      </plugin>
```

从更新JUnit 3.8.1到最新的 4.11。

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

**Maven** 坐标 上面的XML代码片段被称为“Maven坐标”，如果你需要 JUnit 的 jar，你需要找出其相应的 Maven 坐标。它适用于所有其他的依赖，如Spring，Hibernate，Apache 普通的等，只要到Maven中心储存库，并找出哪些是依赖正确的 Maven 坐标。pom.xml – 更新版本

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mkyong</groupId>
    <artifactId>NumberGenerator</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>NumberGenerator</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
      <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
      </plugins>
    </build>

</project>
```

在终端，再次发出同样的命令 mvn eclipse:eclipse ,Maven将从Maven中心储存库下载插件项目依赖关系（JUnit），它会自动保存到你的本地仓库。

## 5. 更新业务逻辑

测试驱动开发（TDD），先更新单元测试，以确保应用程序（APP）对象有一个方法来生成包含恰好36位字母表的唯一密钥。

AppTest.java

```
package com.yiibai;

import org.junit.Assert;
import org.junit.Test;

public class AppTest {

    @Test
    public void testLengthOfTheUniqueKey() {

        App obj = new App();
        Assert.assertEquals(36, obj.generateUniqueKey().length());

    }
}
```

完成业务逻辑。

App.java

```
package com.yiibai;

import java.util.UUID;

/**
 * Generate a unique number
 *
 */
public class App
{

    public static void main( String[] args )
    {
        App obj = new App();
        System.out.println("Unique ID : " + obj.generateUniqueKey(
    }

    public String generateUniqueKey(){

        String id = UUID.randomUUID().toString();
        return id;

    }
}
```

# 6. Maven 打包

现在，我们将使用Maven这个项目，并输出编译成一个 “jar” 的文件。 请参考 pom.xml 文件，包元素定义应该包应该输出什么。

pom.xml

```
<project ...>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.yiibai</groupId>
    <artifactId>NumberGenerator</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
```

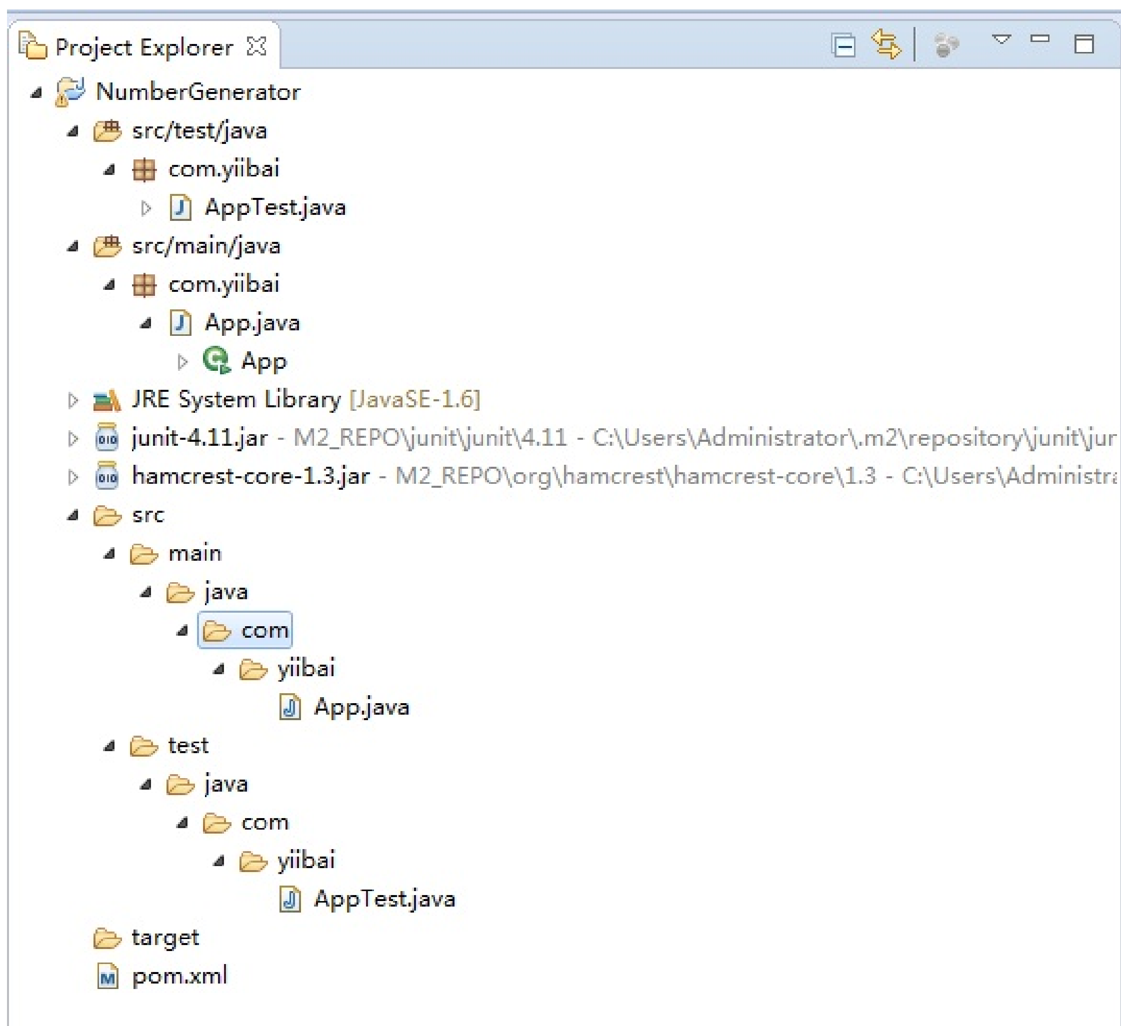
在终端输入 mvn package :

```
C:\worksp\NumberGenerator> mvn package
... ...
ha-2/classworlds-1.1-alpha-2.jar (37 KB at 20.2 KB/sec)
Downloaded: [https://repo.maven.apache.org/maven2/org/codehaus/ple
Downloaded: [https://repo.maven.apache.org/maven2/org/codehaus/ple
Downloaded: [https://repo.maven.apache.org/maven2/org/codehaus/ple
Downloaded: [https://repo.maven.apache.org/maven2/org/codehaus/ple
[INFO] Building jar: C:\worksp\NumberGenerator\target\NumberGenera
OT.jar
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 01:00 min
[INFO] Finished at: 2015-10-27T20:00:17+08:00
[INFO] Final Memory: 10M/54M
[INFO] ------------------------------------------------------------
```

它编译，运行单元测试并打包项目成一个 jar 文件，并把它放在 project/target 文件夹。如果出错：**ERROR: Unable to locate the Javac Compiler in:, C:\Program Files (x86)\Java\jre6..\lib\tools.jar,Please ensure you are using JDK 1.4 or above and,not a JRE (the com.sun.tools.javac.Main class is required)...**

参考：http://my.oschina.net/u/1449336/blog/199802

最终项目的目录结构, 如下图片：






# 7. 示例

从项目的 jar 文件运行应用程序示例

```
C:\worksp\NumberGenerator>java -cp target/NumberGenerator-1.0-SNAPS
iibai.App
Unique ID : 94e5fd1a-c038-415f-a8ed-7fc58c397369
C:\worksp\NumberGenerator>
C:\worksp\NumberGenerator>java -cp target/NumberGenerator-1.0-SNAPS
iibai.App
Unique ID : 48df568a-4b4b-4964-b767-664e206ca4b5
C:\worksp\NumberGenerator>java -cp target/NumberGenerator-1.0-SNAPS
iibai.App
Unique ID : 4ac9156c-2e4a-45f4-8644-0707ae28d5a6
```

# 下载代码

下载代码 - Maven-NumberGenerator.zip

# 使用Maven创建Web应用程序项目 - Maven教程

在本教程中，我们将演示如何使用 Maven 创建一个 Java Web 项目(Spring MVC)。

用到的技术/工具：

1. Maven 3.3.3
2. Eclipse 4.3
3. JDK 8
4. Spring 4.1.1.RELEASED
5. Tomcat 7
6. Logback 1.0.13

## 1. 从Maven模板创建Web项目

您可以通过使用Maven的maven-archetype-webapp模板来创建一个快速启动Java Web应用程序的项目。在终端(* UNIX或Mac)或命令提示符(Windows)中，导航至您想要创建项目的文件夹。

键入以下命令：

```
$ mvn archetype:generate -DgroupId=com.yiibai -DartifactId=CounterW
```

具体示例 ：

```
C:\worksp>mvn archetype:generate -DgroupId=com.yiibai  -DartifactI
p -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=fa
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > gene
@ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < gene
@ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ star
-
[INFO] Generating project in Batch mode
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/
ven-archetype-webapp/1.0/maven-archetype-webapp-1.0.jar
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/a
en-archetype-webapp/1.0/maven-archetype-webapp-1.0.jar (4 KB at 0.1
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/
ven-archetype-webapp/1.0/maven-archetype-webapp-1.0.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/a
en-archetype-webapp/1.0/maven-archetype-webapp-1.0.pom (533 B at 0
[INFO] ------------------------------------------------------------
---
[INFO] Using following parameters for creating project from Old (1
 maven-archetype-webapp:1.0
[INFO] ------------------------------------------------------------
---
[INFO] Parameter: basedir, Value: C:\worksp
[INFO] Parameter: package, Value: com.yiibai
[INFO] Parameter: groupId, Value: com.yiibai
[INFO] Parameter: artifactId, Value: CounterWebApp
[INFO] Parameter: packageName, Value: com.yiibai
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\worksp\C
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 10:30 min
[INFO] Finished at: 2015-10-28T20:31:03+08:00
[INFO] Final Memory: 16M/174M
[INFO] ------------------------------------------------------------
```

新的Web项目命名为 “CounterWebApp”，以及一些标准的 web 目录结构也会自动创建。

# 2. 项目目录布局

查看生成的项目结构布局：

```
.|____CounterWebApp
||____pom.xml
||____src
|||____main
||||____resources
||||____webapp
|||||____index.jsp
|||||____WEB-INF
||||||____web.xml
```

Maven 产生了一些文件夹，一个部署描述符 web.xml，pom.xml 和 index.jsp。

注意， 请查看官方Maven标准目录布局指南来了解更多。

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.yiibai</groupId>
  <artifactId>CounterWebApp</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>CounterWebApp Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>CounterWebApp</finalName>
  </build>
</project>
```

web.xml – Servlet 2.3 已经比较旧, 建议升级到2.5

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" ><web-app><display-name>
```

index.jsp – 一个简单的 hello world html 页面文件

```
<html>
<body>
<div><div class="ads-in-post hide_if_width_less_800">
<script async src="//pagead2.googlesyndication.com/pagead/js/adsbyg
<!-- 728x90 - After2ndH4 -->
<ins class="adsbygoogle hide_if_width_less_800"
     style="display:inline-block;width:728px;height:90px"
     data-ad-client="ca-pub-2836379775501347"
     data-ad-slot="3642936086"
     data-ad-region="mkyongregion"></ins>
<script>
(adsbygoogle = window.adsbygoogle || []).push({});
</script>
</div></div><h2>Hello World!</h2>
</body>
</html>
```

# 3. Eclipse IDE 支持

要导入这个项目到Eclipse中，需要生成一些 Eclipse 项目的配置文件：

3.1、在终端，进入到 “CounterWebApp” 文件夹中，键入以下命令：

```
C:\worksp>cd CounterWebApp
C:\worksp\CounterWebApp>mvn eclipse:eclipse -Dwtpversion=2.0
[INFO] Scanning for projects...
Downloading: [https://repo.maven.apache.org/maven2/org/apache/maven
Downloaded: [https://repo.maven.apache.org/maven2/org/apache/maven/
Downloading: [https://repo.maven.apache.org/maven2/org/apache/maven
Downloaded: [https://repo.maven.apache.org/maven2/org/apache/maven/
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building CounterWebApp Maven Webapp 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.10:eclipse (default-cli) > genera
@ CounterWebApp >>>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.10:eclipse (default-cli) < genera
@ CounterWebApp <<<
[INFO]
[INFO] --- maven-eclipse-plugin:2.10:eclipse (default-cli) @ Counte
[INFO] Adding support for WTP version 2.0.
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launchin
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "CounterWebApp" to C:\worksp\Count
[INFO]
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 7.982 s
[INFO] Finished at: 2015-10-28T20:24:57+08:00
[INFO] Final Memory: 15M/146M
[INFO] ------------------------------------------------------------
```

注意，此选项 -Dwtpversion=2.0 告诉 Maven 将项目转换到 Eclipse 的 Web 项目(WAR)，而不是默认的Java项目(JAR)。为方便起见，以后我们会告诉你如何配置 pom.xml 中的这个 WTP 选项。

3.2 导入到 Eclipse IDE – File -> Import… -> General -> Existing Projects into workspace.



图像说明*: 在 Eclipse* 中，如果看到项目顶部有地球图标，意味着这是一个 *Web* 项目。

# 4. 更新POM

在Maven中，Web项目的设置都通过这个单一的pom.xml文件配置。

1. 添加项目依赖 - Spring, logback 和 JUnit
2. 添加插件来配置项目

阅读注释清楚明了。

pom.xml

```
<project xmlns="[http://maven.apache.org/POM/4.0.0](http://maven.ap
    xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance](http://w
    xsi:schemaLocation="[http://maven.apache.org/POM/4.0.0](http:/
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.yiibai</groupId>
    <artifactId>CounterWebApp</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>CounterWebApp Maven Webapp</name>
    <url>[http://maven.apache.org</url>](http://maven.apache.org%3C
        <jdk.version>1.7</jdk.version>
        <spring.version>4.1.1.RELEASE</spring.version>
        <jstl.version>1.2</jstl.version>
        <junit.version>4.11</junit.version>
        <logback.version>1.0.13</logback.version>
        <jcl-over-slf4j.version>1.7.5</jcl-over-slf4j.version>
    </properties>

    <dependencies>
        <!-- Unit Test -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
        </dependency>

        <!-- Spring Core -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${jcl-over-slf4j.version}</version>
        </dependency>
        <dependency>
```

```
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>${logback.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <!-- jstl -->
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>${jstl.version}</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>CounterWebApp</finalName>

        <plugins>
            <!-- Eclipse project -->
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-eclipse-plugin</artifactId>
            <version>2.9</version>
            <configuration>
                    <!-- Always download and attach dependencies sc
                <downloadSources>true</downloadSources>
                <downloadJavadocs>false</downloadJavadocs>
                <!-- Avoid type mvn eclipse:eclipse -Dwtpversion=2.
                <wtpversion>2.0</wtpversion>
            </configuration>
          </plugin>

          <!-- Set JDK Compiler Level -->
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>${jdk.version}</source>
                <target>${jdk.version}</target>
            </configuration>
          </plugin>
```

```
            <!-- For Maven Tomcat Plugin -->
            <plugin>
              <groupId>org.apache.tomcat.maven</groupId>
              <artifactId>tomcat7-maven-plugin</artifactId>
              <version>2.2</version>
              <configuration>
                  <path>/CounterWebApp</path>
              </configuration>
            </plugin>
          </plugins>
      </build>
</project>
```

注意，为方便起见，声明 maven-eclipse-plugin，并配置wtpversion 来避免输入参数 -Dwtpversion=2.0。现在，每次使用 mvn eclipse:eclipse，Maven将这个项目导入转换为 Eclipse Web 项目。

```
#之前
mvn eclipse:eclipse --> Eclipse Java project (JAR)
mvn eclipse:eclipse -Dwtpversion=2.0 --> Eclipse Java web project (

#之后
mvn eclipse:eclipse --> Eclipse Java web project (WAR)
```

# 5. 更新源代码

在这一步中，在上一步配置完pom.xml后，重新执行 mvn eclipse:eclipse 这个命令，我们将创建Spring MVC的一些文件和logback日志框架的文件夹，最终的项目结构如下所示：

```
.
|____pom.xml
|____src
| |____main
| | |____java
| | | |____com
| | | | |____yiibai
| | | | | |____controller
| | | | | | |____BaseController.java
| | |____resources
| | | |____logback.xml
| | |____webapp
| | | |____WEB-INF
| | | | |____mvc-dispatcher-servlet.xml
| | | | |____pages
| | | | | |____index.jsp
| | | | |____web.xml
```

注意，如果它不存在，需要手动创建文件夹。

5.1 创建 Spring MVC 的控制器类。

/src/main/java/com/yiibai/controller/BaseController.java

```
package com.yiibai.controller;

import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class BaseController {

    private static int counter = 0;
    private static final String VIEW_INDEX = "index";
    private final static org.slf4j.Logger logger = LoggerFactory.ge

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String welcome(ModelMap model) {

        model.addAttribute("message", "Welcome");
        model.addAttribute("counter", ++counter);
        logger.debug("[welcome] counter : {}", counter);

        // Spring uses InternalResourceViewResolver and return back
        return VIEW_INDEX;

    }

    @RequestMapping(value = "/{name}", method = RequestMethod.GET)
    public String welcomeName(@PathVariable String name, ModelMap m

        model.addAttribute("message", "Welcome " + name);
        model.addAttribute("counter", ++counter);
        logger.debug("[welcomeName] counter : {}", counter);
        return VIEW_INDEX;

    }

}
```

5.2 创建Spring配置文件。

/src/main/webapp/WEB-INF/mvc-dispatcher-servlet.xml

```
<beans xmlns="[http://www.springframework.org/schema/beans](http:/
    xmlns:context="[http://www.springframework.org/schema/context](
    xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance](http://w
    xsi:schemaLocation="
        [http://www.springframework.org/schema/beans](http://www.sp
        [http://www.springframework.org/schema/beans/spring-beans.x
        [http://www.springframework.org/schema/context/spring-conte

    <context:component-scan base-package="com.yiibai.controller" />
    <bean
        class="org.springframework.web.servlet.view.InternalResourc
        <property name="prefix">
            <value>/WEB-INF/pages/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>

</beans>
```

5.3 更新让现有的 web.xml 支持 Servlet 2.5(默认的Servlet2.3 太旧了)， 并且还通过 Spring 监听器 ContextLoaderListener 集成了Spring框架。

/src/main/webapp/WEB-INF/web.xml

```
<web-app xmlns="[http://java.sun.com/xml/ns/javaee](http://java.sun
        xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance](http
    xsi:schemaLocation="[http://java.sun.com/xml/ns/javaee](http:/
          [http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd](http
    version="2.5">

    <display-name>Counter Web Application</display-name>

    <servlet>
        <servlet-name>mvc-dispatcher</servlet-name>
        <servlet-class>
                        org.springframework.web.servlet.DispatcherS
                </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>mvc-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-val
    </context-param>

    <listener>
        <listener-class>
                   org.springframework.web.context.ContextLoaderLis
                </listener-class>
    </listener>
</web-app>
```

5.4 移动文件 index.jsp 到 WEB-INF/pages 目录下， 为了保护直接访问。并更新内容：

/src/main/webapp/WEB-INF/pages/index.jsp

5.5 在资源文件夹(resources)中创建 logback.xml 文件

/src/main/resources/logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

# 6. Eclipse + Tomcat

在第5步中创建所有文件以后，这里有一些方法可以用来部署和测试Web项目，我们这里推荐使用6.2中的方法。

6.1 要编译，测试和项目打包成一个WAR文件，输入：

```
mvn package
```

一个新的 WAR 文件将在 project/target/CounterWebApp.war产生，只需复制并部署到Tomcat 发布的目录。

6.2 如果想通过 Eclipse 服务器这个项目插件(Tomcat 或其它容器)调试，这里再输入：

```
mvn eclipse:eclipse
```

如果一切顺利，该项目的依赖将被装配附加到 Web部署项目。图片*: 右键点击 project -> Properties -> Deployment Assembly*

6.3 Maven 的 Tomcat 插件声明(加入到 pom.xml)：

pom.xml

```
<!-- For Maven Tomcat Plugin -->
```

键入以下命令(有时网络不通畅需要执行2-3次)：

```
mvn tomcat:run
tp://logback.qos.ch/codes.html#layoutInsteadOfEncoder for details
20:37:32,089 |-INFO in ch.qos.logback.classic.joran.action.LoggerAc
g level of logger [com.yiibai.controller] to DEBUG
20:37:32,089 |-INFO in ch.qos.logback.classic.joran.action.LoggerAc
g additivity of logger [com.yiibai.controller] to false
20:37:32,090 |-INFO in ch.qos.logback.core.joran.action.AppenderRef
ching appender named [STDOUT] to Logger[com.yiibai.controller]
20:37:32,090 |-INFO in ch.qos.logback.classic.joran.action.RootLogg
tting level of ROOT logger to ERROR
20:37:32,090 |-INFO in ch.qos.logback.core.joran.action.AppenderRef
ching appender named [STDOUT] to Logger[ROOT]
20:37:32,090 |-INFO in ch.qos.logback.classic.joran.action.Configur
 End of configuration.
20:37:32,091 |-INFO in [ch.qos.logback.classic.joran.JoranConfigura

十月 28, 2015 20:37:32 下午 org.apache.catalina.core.ApplicationCont
信息: Initializing Spring root WebApplicationContext
十月 28, 2015 20:37:33 下午 org.apache.catalina.core.ApplicationCont
信息: Initializing Spring FrameworkServlet 'mvc-dispatcher'
十月 28, 2015 20:37:33 下午 org.apache.coyote.http11.Http11Protocol
信息: Initializing Coyote HTTP/1.1 on http-8080
十月 28, 2015 20:37:33 下午 org.apache.coyote.http11.Http11Protocol
信息: Starting Coyote HTTP/1.1 on http-8080
```

这将启动Tomcat，部署项目默认在端口8080。

出错：Maven项目下update maven后Eclipse报错：java.lang.ClassNotFoundException: ContextLoaderL

解决方案：

**1.**右键点击项目**--**选择**Properties**

选择Deployment Assembly,在右边点击Add按钮，在弹出的窗口中选择Java Build Path Entries

**2.**点击**Next**，选择Maven Dependencies

**3.**点击**Finish**，然后可以看到已经把Maven Dependencies添加到Web应用结构中了

操作完后，重新部署工程，不再报错了。然后我们再到.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\目录下，发现工程WEB-INF目录下自动生成了lib目录，并且所有的依赖jar包也都已经部署进来。问题因此解决。

# Maven外部依赖 - Maven教程

现在，你也知道Maven做依赖管理使用Maven仓库的概念。但是，如果依赖是不提供任何远程存储库和中央存储库发生了什么？ Maven提供为使用外部依赖的概念，应用在这样的场景。

举一个例子，让我们做以下修改项在目Maven创建项目 这一章节中创建。

- 添加lib文件夹到src文件夹
- 复制任何的jar到lib文件夹。我们使用ldapjdk.jar，这是LDAP操作的辅助库。

现在我们的项目结构看起来应该像下面的：




在这里，有自己的特定项目，这是很平常案例库，它可以包含jar文件可能无法在任何Maven存储库，那么需要下载。如果代码使用这个Maven库，那么Maven构建将失败，因为它无法下载或在编译阶段是指这个库。

要处理这种情况，让我们来添加这个外部依赖项中使用下列方式到Maven的pom.xml。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.companyname.bank</groupId>
   <artifactId>consumerBanking</artifactId>
   <packaging>jar</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>consumerBanking</name>
   <url>http://maven.apache.org</url>

   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
      </dependency>

      <dependency>
         <groupId>ldapjdk</groupId>
         <artifactId>ldapjdk</artifactId>
         <scope>system</scope>
         <version>1.0</version>
         <systemPath>${basedir}srclibldapjdk.jar</systemPath>
      </dependency>
   </dependencies>

</project>
```

先看下依赖性在上面的例子，清除下列有关外部相关的关键概念第二dependency元素。

- 外部依赖（jar库位置）可以在pom.xml中以同样的方式与其他依赖关系进行配置。
- 指定的groupId一样的库名称。
- 指定artifactId的相同库的名称。
- 指定范围的系统。
- 指定相系统项目的位置。

希望现在你清楚了解外部依赖，能够指定在Maven项目的外部依赖。

# Maven项目文档 - Maven教程

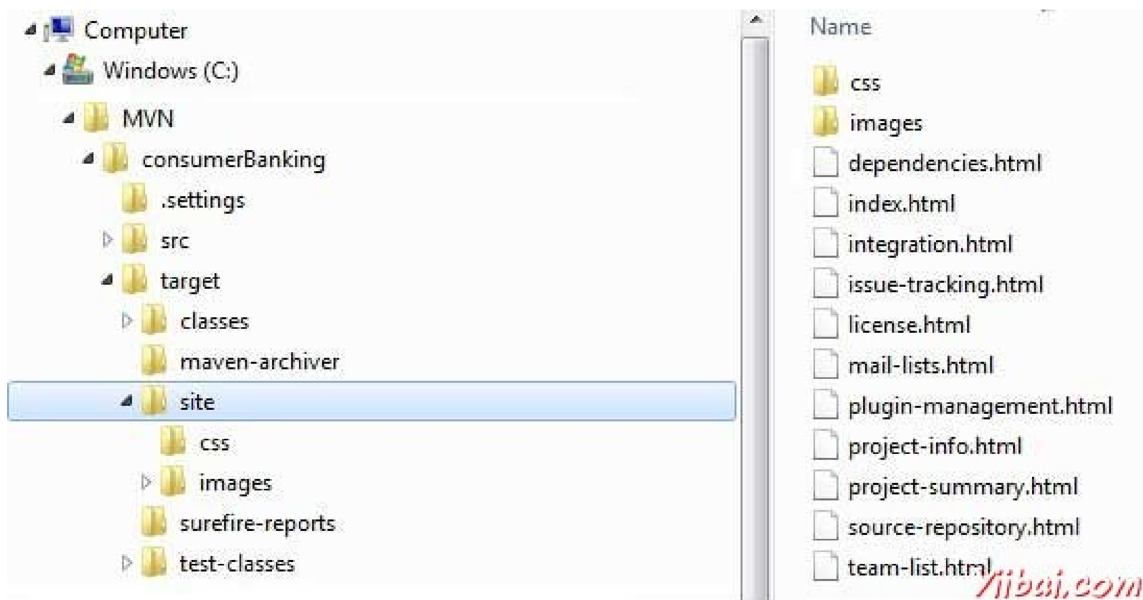
本教程将教你如何一步到位创建应用程序的文档。因此，让我们开始，到 C:/MVN 创建java应用程序consumerBanking。 OpenconsumerBanking文件夹，然后执行以下命令mvn命令。

```
C:MVN>mvn site
```

Maven将开始构建该项目。

```
[INFO] Scanning for projects...
[INFO] -------------------------------------------------------------------
[INFO] Building consumerBanking
[INFO]    task-segment: [site]
[INFO] -------------------------------------------------------------------
[INFO] [site:site {execution: default-site}]
[INFO] artifact org.apache.maven.skins:maven-default-skin:
checking for updates from central
[INFO] Generating "About" report.
[INFO] Generating "Issue Tracking" report.
[INFO] Generating "Project Team" report.
[INFO] Generating "Dependencies" report.
[INFO] Generating "Continuous Integration" report.
[INFO] Generating "Source Repository" report.
[INFO] Generating "Project License" report.
[INFO] Generating "Mailing Lists" report.
[INFO] Generating "Plugin Management" report.
[INFO] Generating "Project Summary" report.
[INFO] -------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -------------------------------------------------------------------
[INFO] Total time: 16 seconds
[INFO] Finished at: Wed Jul 11 18:11:18 IST 2012
[INFO] Final Memory: 23M/148M
[INFO] -------------------------------------------------------------------
```

就是这样。你的项目文件已准备就绪。 Maven有目标目录中创建一个网站。




打开 C:MVNconsumerBanking argetsite 文件夹。点击index.htmll看到文档。

# consumerBanking

Last Published: 2012-07-11 consumerBanking

Project Documentation
- Project Information
  - About
  - Continuous Integration
  - Dependencies
  - Issue Tracking
  - Mailing Lists
  - Plugin Management
  - Project License
  - **Project Summary**
  - Project Team
  - Source Repository

Built by: maven

## Project Summary

### Project Information

| Field | Value |
| --- | --- |
| Name | consumerBanking |
| Description | - |
| Homepage | http://maven.apache.org |

### Project Organization

This project does not belong to an organization.

### Build Information

| Field | Value |
| --- | --- |
| GroupId | com.companyname.bank |
| ArtifactId | consumerBanking |
| Version | 1.0-SNAPSHOT |
| Type | jar |

Yiibai.com

Maven会使用称为Doxia一个文件处理引擎，它会读取多个源格式转换为通用文档模型的文档。要编写你的项目文档，可以在以下几个常用的格式，这是由Doxia解析编写内容。

| 格式名称 | 描述 | 参考 |
| --- | --- | --- |
| APT | A Plain Text document format | http://maven.apache.org/doxia/format.htmll |
| XDoc | A Maven 1.x documentation format | http://jakarta.apache.org/site/jakarta-site2.htmll |
| FML | Used for FAQ documents | http://maven.apache.org/doxia/references/fml-format.htmll |
| XHTML | Extensible HTML | http://en.wikipedia.org/wiki/XHTML |

# Maven项目模板 - Maven教程

Maven提供用户，使用原型的概念，不同类型的项目模板（以数字614）是一个非常大的列表。 Maven帮助用户快速开始使用以下命令创建新的Java项目

```
mvn archetype:generate
```

## 什么是原型？

原型是一个Maven插件，其任务是创建一个项目结构按照其模板。我们将使用快速启动原型插件在这里创建一个简单的Java应用程序。

## 使用项目模板

让我们打开命令控制台，进入到C：> MVN目录，然后执行以下命令mvn命令

```
C:MVN>mvn archetype:generate
```

Maven会开始处理，并会要求选择所需的原型

```
INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] -----------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [archetype:generate] (aggregator-style)
[INFO] -----------------------------------------------------------
[INFO] Preparing archetype:generate
...
600: remote -> org.trailsframework:trails-archetype (-)
601: remote -> org.trailsframework:trails-secure-archetype (-)
602: remote -> org.tynamo:tynamo-archetype (-)
603: remote -> org.wicketstuff.scala:wicket-scala-archetype (-)
604: remote -> org.wicketstuff.scala:wicketstuff-scala-archetype
Basic setup for a project that combines Scala and Wicket,
depending on the Wicket-Scala project.
Includes an example Specs test.)
605: remote -> org.wikbook:wikbook.archetype (-)
606: remote -> org.xaloon.archetype:xaloon-archetype-wicket-jpa-gla
607: remote -> org.xaloon.archetype:xaloon-archetype-wicket-jpa-spr
608: remote -> org.xwiki.commons:xwiki-commons-component-archetype
(Make it easy to create a maven project for creating XWiki Componen
609: remote -> org.xwiki.rendering:xwiki-rendering-archetype-macro
(Make it easy to create a maven project for creating XWiki Renderin
610: remote -> org.zkoss:zk-archetype-component (The ZK Component a
611: remote -> org.zkoss:zk-archetype-webapp (The ZK wepapp archety
612: remote -> ru.circumflex:circumflex-archetype (-)
613: remote -> se.vgregion.javg.maven.archetypes:javg-minimal-arche
614: remote -> sk.seges.sesam:sesam-annotation-archetype (-)
Choose a number or apply filter
(format: [groupId:]artifactId, case sensitive contains): 203:
```

按Enter键选择默认选项（203：Maven原型 - 快速入门）

Maven会要求原型的特定版本

```
Choose org.apache.maven.archetypes:maven-archetype-quickstart versi
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: 6:
```

按Enter键选择默认选项（6：Maven原型 - 快速入门：1.1）

Maven会要求项目的细节。输入项目细节的要求。按回车，如果提供的默认值。您可以通过输入自己的值覆盖它们。

```
Define value for property 'groupId': : com.companyname.insurance
Define value for property 'artifactId': : health
Define value for property 'version': 1.0-SNAPSHOT:
Define value for property 'package': com.companyname.insurance:
```

Maven会要求项目的细节确认。按回车键或按Y

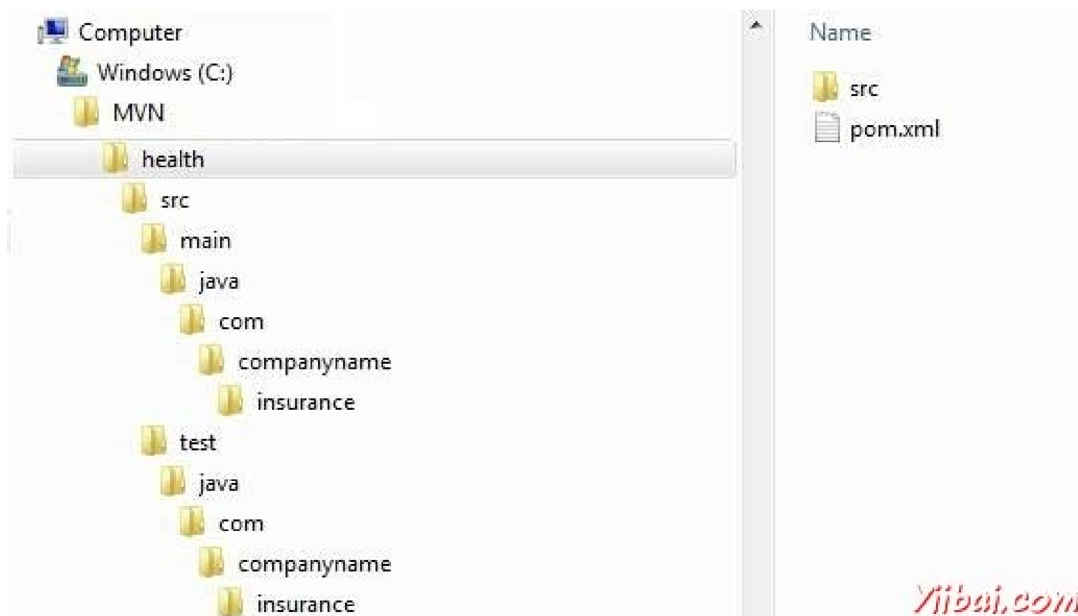
```
Confirm properties configuration:
groupId: com.companyname.insurance
artifactId: health
version: 1.0-SNAPSHOT
package: com.companyname.insurance
Y:
```

现在，Maven将开始创建项目结构，并会显示如下内容：

```
[INFO] -----------------------------------------------------------------
[INFO] Using following parameters for creating project
from Old (1.x) Archetype: maven-archetype-quickstart:1.1
[INFO] -----------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.companyname.insurance
[INFO] Parameter: packageName, Value: com.companyname.insurance
[INFO] Parameter: package, Value: com.companyname.insurance
[INFO] Parameter: artifactId, Value: health
[INFO] Parameter: basedir, Value: C:MVN
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:MVNhealth
[INFO] -----------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -----------------------------------------------------------------
[INFO] Total time: 4 minutes 12 seconds
[INFO] Finished at: Fri Jul 13 11:10:12 IST 2012
[INFO] Final Memory: 20M/90M
[INFO] -----------------------------------------------------------------
```

# 创建项目

现在转到C：> MVN目录。会看到一个java应用程序项目创建了这是在创建项目时给出 artifactId 命名为：health 。 Maven将创建一个标准的目录布局如下图所示的项目：




## 创建的pom.xml

Maven生成如下所列项目中的 pom.xml 文件：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.companyname.insurance</groupId>
   <artifactId>health</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>health</name>
   <url>http://maven.apache.org</url>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEnc
   </properties>
   <dependencies>
      <dependency>
      <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
</project>
```

## 创建App.java

Maven 示例生成Java源文件，App.java下面列出项目：

位置: **C: > MVN > health > src > main > java > com > companyname > insurance > App.java**

```
package com.companyname.insurance;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

# 创建**AppTest.java**

Maven的样本生成Java源测试文件，AppTest.java下面列出的项目：

位置: **C: > MVN > health > src > test > java > com > companyname > insurance > AppTest.java**

```
package com.companyname.insurance;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Unit test for simple App.
 */
public class AppTest
    extends TestCase
{
    /**
     * Create the test case
     *
     * @param testName name of the test case
     */
    public AppTest( String testName )
    {
        super( testName );
    }

    /**
     * @return the suite of tests being tested
     */
    public static Test suite()
    {
        return new TestSuite( AppTest.class );
    }

    /**
     * Rigourous Test :-)
     */
    public void testApp()
    {
        assertTrue( true );
    }
}
```

就是这样。现在可以看到Maven的功能。您可以创建任何类型，使用 maven 单一命令的项目并启动开发。

# Maven快照 - Maven教程

大型应用软件一般由多个模块，它是多个团队正在开发同一个应用程序的不同模块，其中常见的场景。例如，考虑一个团队正在对应用程序的应用程序，用户界面项目(app-ui.jar:1.0) 的前端和他们正在使用的数据服务计划 (data-service.jar:1.0)。

现在，它可能发生，团队工作的数据服务正在发生快速的步伐bug修复或增强功能和它们释放出库到远程仓库几乎每隔一天。

现在，如果数据服务团队上传新版本隔日然后会出现下面的问题

- 数据服务的团队应该每次都告诉应用程序UI的团队时，他们已经发布了更新后的代码。
- UI团队需要经常更新自己的pom.xml中获得更新的版本的应用程序。

为了处理这类情况，快照的概念开始发挥作用。

## 什么是快照？

快照是一个特殊版本，指出目前开发复印件。不同于常规版本，Maven的检查新的快照版本中，每生成一个远程存储库。

现在，数据服务团队将公布更新后的代码每次的快照存储库说，数据服务:1.0-SNAPSHOT替换一个旧的SNAPSHOT jar。

## 快照与版本

如遇版本的，如果一旦Maven的下载所提到的版本为，data-service:1.0，它永远不会尝试下载更新1.0可在库中。要下载更新的代码，数据服务版本升级到1.1。

Maven会自动获取最新的快照（data-service:1.0-SNAPSHOT）每次应用程序UI团队建立自己的项目。

## app-ui pom.xml

app-ui 项目使用数据服务的1.0-SNAPSHOT

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>app-ui</groupId>
   <artifactId>app-ui</artifactId>
   <version>1.0</version>
   <packaging>jar</packaging>
   <name>health</name>
   <url>http://maven.apache.org</url>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEnc
   </properties>
   <dependencies>
      <dependency>
      <groupId>data-service</groupId>
         <artifactId>data-service</artifactId>
         <version>1.0-SNAPSHOT</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
</project>
```

## data-service pom.xml

数据服务项目释放1.0快照对于每一个微小的变化

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>data-service</groupId>
   <artifactId>data-service</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>health</name>
   <url>http://maven.apache.org</url>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEnc
   </properties>
   </project>
```

虽然，如快照，Maven自动获取上每天最新的快照。您可以强制使用-U切换到任何maven命令来下载最新的快照版本。

```
mvn clean package -U
```

让我们打开命令控制台，进入到 **C: > MVN > app-ui** 目录，然后执行以下命令mvn命令。

```
C:MVNapp-ui>mvn clean package -U
```

Maven会下载数据服务的最新快照后开始构建该项目。

```
[INFO] Scanning for projects...
[INFO] -----------------------------------------------------------------
[INFO] Building consumerBanking
[INFO]    task-segment: [clean, package]
[INFO] -----------------------------------------------------------------
[INFO] Downloading data-service:1.0-SNAPSHOT
[INFO] 290K downloaded.
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting directory C:MVNapp-ui    arget
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (Cp1252 actually) to copy filtere
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:MVNapp-uisrcmain
resources
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Compiling 1 source file to C:MVNapp-ui    argetclasses
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (Cp1252 actually) to copy filtere
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:MVNapp-uisrc    est
resources
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Compiling 1 source file to C:MVNapp-ui    arget    est-class
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: C:MVNapp-ui    arget
surefire-reports
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.companyname.bank.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.0

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: C:MVNapp-ui    arget
app-ui-1.0-SNAPSHOT.jar
[INFO] -----------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -----------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Tue Jul 10 16:52:18 IST 2012
[INFO] Final Memory: 16M/89M
[INFO] -----------------------------------------------------------------
```

# Maven构建自动化-Hudson - Maven教程

建立自动化定义场景，依赖项目建设过程中被启动，一旦项目生成成功完成，以确保相关的项目是稳定的。

## 实例

考虑一个团队正在开发一个项目总线核心API上的其他两个项目的应用程序，网页UI和应用程序的桌面UI的依赖。

app-web-ui 项目使用1.0-SNAPSHOT总线核心API项目

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>app-web-ui</groupId>
   <artifactId>app-web-ui</artifactId>
   <version>1.0</version>
   <packaging>jar</packaging>
   <dependencies>
      <dependency>
      <groupId>bus-core-api</groupId>
         <artifactId>bus-core-api</artifactId>
         <version>1.0-SNAPSHOT</version>
      </dependency>
   </dependencies>
</project>
```

app-desktop-ui 项目使用总线核心API项目的1.0-SNAPSHOT

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>app-desktop-ui</groupId>
   <artifactId>app-desktop-ui</artifactId>
   <version>1.0</version>
   <packaging>jar</packaging>
   <dependencies>
      <dependency>
      <groupId>bus-core-api</groupId>
         <artifactId>bus-core-api</artifactId>
         <version>1.0-SNAPSHOT</version>
      </dependency>
   </dependencies>
</project>
```

bus-core-api 项目

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>bus-core-api</groupId>
   <artifactId>bus-core-api</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
</project>
```

现在，app-web-ui和app-desktop-ui项目团队需要自己编译过程应该揭开序幕，每当bus-core-api项目的变化。

使用快照确保应使用最新的bus-core-api 项目，但要满足上面我们需要做一些额外的要求。

- 添加一个生成后的目标bus-core-api POM的应用程序 app-web-ui 和app-desktop-ui 的基础之上。
- 使用持续集成（CI）的服务器像哈德森自动管理构建自动化。

# 使用Maven

更新总线核心API项目pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>bus-core-api</groupId>
   <artifactId>bus-core-api</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <build>
   <plugins>
   <plugin>
   <artifactId>maven-invoker-plugin</artifactId>
   <version>1.6</version>
      <configuration>
         <debug>true</debug>
         <pomIncludes>
            <pomInclude>app-web-ui/pom.xml</pomInclude>
            <pomInclude>app-desktop-ui/pom.xml</pomInclude>
         </pomIncludes>
      </configuration>
      <executions>
         <execution>
            <id>build</id>
            <goals>
               <goal>run</goal>
            </goals>
         </execution>
      </executions>
   </plugin>
   </plugins>
   <build>
</project>
```

让我们打开命令控制台，进入到C: > MVN > bus-core-api目录，然后执行以下命令mvn命令。

```
C:MVNus-core-api>mvn clean package -U
```

Maven将开始构建项目bus-core-api。

```
[INFO] Scanning for projects...
[INFO] -------------------------------------------------------------------
[INFO] Building bus-core-api
[INFO]    task-segment: [clean, package]
[INFO] -------------------------------------------------------------------
...
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: C:MVNus-core-ui    arget
bus-core-ui-1.0-SNAPSHOT.jar
[INFO] -------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -------------------------------------------------------------------
```

一旦bus-core-api构建成功，Maven将开始构建应用程序app-web-ui

```
[INFO] -------------------------------------------------------------------
[INFO] Building app-web-ui
[INFO]    task-segment: [package]
[INFO] -------------------------------------------------------------------
...
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: C:MVNapp-web-ui    arget
app-web-ui-1.0-SNAPSHOT.jar
[INFO] -------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -------------------------------------------------------------------
```

一旦app-web-ui 构建成功，Maven将开始构建app-desktop-ui项目

```
[INFO] -------------------------------------------------------------------
[INFO] Building app-desktop-ui
[INFO]    task-segment: [package]
[INFO] -------------------------------------------------------------------
...
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: C:MVNapp-desktop-ui    arget
app-desktop-ui-1.0-SNAPSHOT.jar
[INFO] -------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -------------------------------------------------------------------
```

# 使用Maven持续集成服务

使用CI服务器更适合作为开发人员不需要更新的bus-core-api I项目的POM每次一个新的项目，例如 app-mobile-ui添加作为bus-core-api 项目相关的项目。哈德森Hudson 自动管理使用Maven的依赖管理构建自动化。

哈德森（Hudson ）认每个项目生成的工作。一旦一个项目的代码签入到SVN（或映射到哈德森任何源管理工具），哈德森开始它的构建工作，一旦这项工作得到完成，它会自动启动其他相关工作（其他相关项目）。

在上面的例子中，当bus-core-ui 源代码SVN更新，哈德森开始它的构建。一旦构建成功。哈德森自动查找相关的项目，并开始构建app-web-ui 和app-desktop-ui 项目。

# Maven依赖管理 - Maven教程

其中一个Maven的核心特征是依赖管理。管理依赖关系变得困难的任务一旦我们处理多模块项目（包含数百个模块/子项目）。 Maven提供了一个高程度的控制来管理这样的场景。

## 传递依赖发现

这是很通常情况下，当一个库说A就依赖于其他库说B的情况下，另一个项目Ç想用A，则该项目需要使用库中B。

在Maven帮助下以避免这样的要求来发现所有需要的库。 Maven通过读取依赖项项目文件（pom.xml中），找出它们的依赖等。

我们只需要在每个项目POM定义直接依赖关系。 Maven自动处理其余部分。

传递依赖，包括库的图形可能会快速增长在很大程度上。可能出现情况下，当有重复的库。 Maven提供一些功能来控制传递依赖程度

| Feature | 描述 |
| --- | --- |
| Dependency mediation | Determines what version of a dependency is to be used when multiple versions of an artifact are encountered. If two dependency versions are at the same depth in the dependency tree, the first declared dependency will be used. |
| Dependency management | Directly specify the versions of artifacts to be used when they are encountered in transitive dependencies. For an example project C can include B as a dependency in its dependencyManagement section and directly control which version of B is to be used when it is ever referenced. |
| Dependency scope | Includes dependencies as per the current stage of the build |
| Excluded dependencies | Any transitive dependency can be excluede using "exclusion" element. As example, A depends upon B and B depends upon C then A can mark C as excluded. |
| Optional dependencies | Any transitive dependency can be marked as optional using "optional" element. As example, A depends upon B and B depends upon C. Now B marked C as optional. Then A will not use C. |

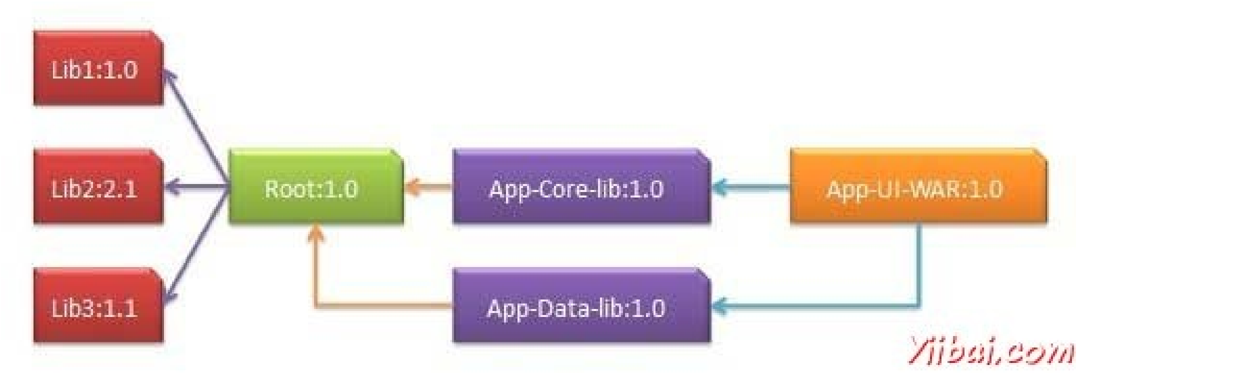
## 依赖范围

传递依赖发现可以使用各种依赖范围如下文所述受到限制

| Scope | 描述 |
| --- | --- |
| compile | This scope indicates that dependency is available in classpath of project. It is default scope. |
| provided | This scope indicates that dependency is to be provided by JDK or web-Server/Container at runtime. |
| runtime | This scope indicates that dependency is not required for compilation, but is required during execution. |
| test | This scope indicates that the dependency is only available for the test compilation and execution phases. |
| system | This scope indicates that you have to provide the system path. |
| import | This scope is only used when dependency is of type pom. This scopes indicates that the specified POM should be replaced with the dependencies in that POM's <dependencyManagement> section. |

# 依赖关系管理

通常情况下，我们已经一套项目在一个共同的项目下。在这种情况下，我们可以创造让所有的公共依赖一个共同的POM，然后进行分项目POMS为这个POM父。下面的例子将帮助你理解这个概念




以下是上述的依赖图的细节

- APP-UI-WAR依赖于App-Core-lib和 App-Data-lib。
- Root 是 App-Core-lib 和 App-Data-lib 的父类。
- Root 定义LIB1，LIB2，Lib3作为其依赖部分依赖关系。

App-UI-WAR

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>App-UI-WAR</artifactId>
      <version>1.0</version>
      <packaging>war</packaging>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname</groupId>
            <artifactId>App-Core-lib</artifactId>
            <version>1.0</version>
         </dependency>
      </dependencies>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname</groupId>
            <artifactId>App-Data-lib</artifactId>
            <version>1.0</version>
         </dependency>
      </dependencies>
</project>
```

App-Core-lib

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <parent>
         <artifactId>Root</artifactId>
         <groupId>com.companyname.groupname</groupId>
         <version>1.0</version>
      </parent>
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>App-Core-lib</artifactId>
      <version>1.0</version>
      <packaging>jar</packaging>
</project>
```

App-Data-lib

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <parent>
         <artifactId>Root</artifactId>
         <groupId>com.companyname.groupname</groupId>
         <version>1.0</version>
      </parent>
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>App-Data-lib</artifactId>
      <version>1.0</version>
      <packaging>jar</packaging>
</project>
```

Root

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>Root</artifactId>
      <version>1.0</version>
      <packaging>pom</packaging>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname1</groupId>
            <artifactId>Lib1</artifactId>
            <version>1.0</version>
         </dependency>
      </dependencies>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname2</groupId>
            <artifactId>Lib2</artifactId>
            <version>2.1</version>
         </dependency>
      </dependencies>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname3</groupId>
            <artifactId>Lib3</artifactId>
            <version>1.1</version>
         </dependency>
      </dependencies>
</project>
```

现在，当我们建立App-UI-WAR项目，Maven会发现所有的依赖通过遍历依赖图和构建应用程序。

从上面的例子中，我们可以学到以下关键概念

- 常见的依赖关系可以用父POM的概念被放置在一个地方。 App-Data-lib 和 App-Core-lib 项目列表在 Root 目录(见Roots包类型。它是POM).
- 不需要Lib1, lib2, Lib3 作为依赖于 App-UI-WAR. Maven使用传递性依赖机制来管理这些细节。

# Maven自动化部署 - Maven教程

在项目开发中，通常是部署过程包含以下步骤

- 检入代码在建项目全部进入SVN或源代码库中，并标记它。
- 从SVN下载完整的源代码。
- 构建应用程序。
- 生成输出要么WAR或EAR文件存储到一个共同的网络位置。
- 从网络获取的文件和文件部署到生产现场。
- 更新日期和应用程序的更新版本号的文件。

## 问题说明

通常有多人参与了上述部署过程。一个团队可能手动签入的代码，其他人可以处理构建等。这很可能是任何一个步骤可能会错过了，由于涉及和由于多团队环境手动工作。例如，较旧的版本可能不会被更换网络设备和部署团队再部署旧版本。

## 解决

通过结合自动化的部署过程

- Maven构建和释放项目，
- SubVersion源代码库，管理源代码，
- 和远程存储库管理器（Jfrog/ Nexus）来管理项目的二进制文件。

## 更新项目的**pom.xml**

我们将使用Maven发布插件来创建一个自动释放过程。

例如：bus-core-api 项目POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>bus-core-api</groupId>
   <artifactId>bus-core-api</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <scm>
      <url>http://www.svn.com</url>
      <connection>scm:svn:http://localhost:8080/svn/jrepo/trunk/
      Framework</connection>
      <developerConnection>scm:svn:${username}/${password}@localhos
      common_core_api:1101:code</developerConnection>
   </scm>
   <distributionManagement>
      <repository>
         <id>Core-API-Java-Release</id>
         <name>Release repository</name>
         <url>http://localhost:8081/nexus/content/repositories/
         Core-Api-Release</url>
      </repository>
   </distributionManagement>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-release-plugin</artifactId>
            <version>2.0-beta-9</version>
            <configuration>
               <useReleaseProfile>false</useReleaseProfile>
               <goals>deploy</goals>
               <scmCommentPrefix>[bus-core-api-release-checkin]-<
               /scmCommentPrefix>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

在pom.xml中，下面是我们使用的重要元素

| 元素 | 描述 |
| --- | --- |
| SCM | Configures the SVN location from where Maven will check out the source code. |
| Repositories | Location where built WAR/EAR/JAR or any other artifact will be stored after code build is successful. |
| Plugin | maven-release-plugin is configured to automate the deployment process. |

## Maven发布插件

Maven使用确实下列有用的任务*maven-release-plugin*.

```
mvn release:clean
```

它清除以防工作区的最后一个释放的过程并不顺利。

```
mvn release:rollback
```

回滚是为了以防工作空间代码和配置更改的最后一个释放的过程并不顺利。

```
mvn release:prepare
```

执行多个操作次数

- 检查是否有任何未提交的本地更改或不
- 确保没有快照依赖
- 更改应用程序的版本并删除快照从版本，以释放
- 更新文件到 SVN.
- 运行测试用例
- 提交修改后POM文件
- 标签代码在subversion中
- 增加版本号和附加快照以备将来发行
- 提交修改后的POM文件到SVN。

```
mvn release:perform
```

检查出使用前面定义的标签代码并运行Maven的部署目标来部署战争或内置工件档案库。

让我们打开命令控制台，到 **C: > MVN >bus-core-api** 目录并执行以下命令mvn命令。

```
C:MVNus-core-api>mvn release:prepare
```

Maven将开始建设该项目。一旦构建成功运行以下命令mvn命令。

```
C:MVNus-core-api>mvn release:perform
```

一旦构建成功，您可以在资料库验证上传的JAR文件。

# Maven Web应用 - Maven教程

本教程将教你如何管理使用Maven版本控制系统管理一个基于Web项目。在这里，将学习如何创建/构建/部署和运行Web应用程序：

## 创建Web应用程序

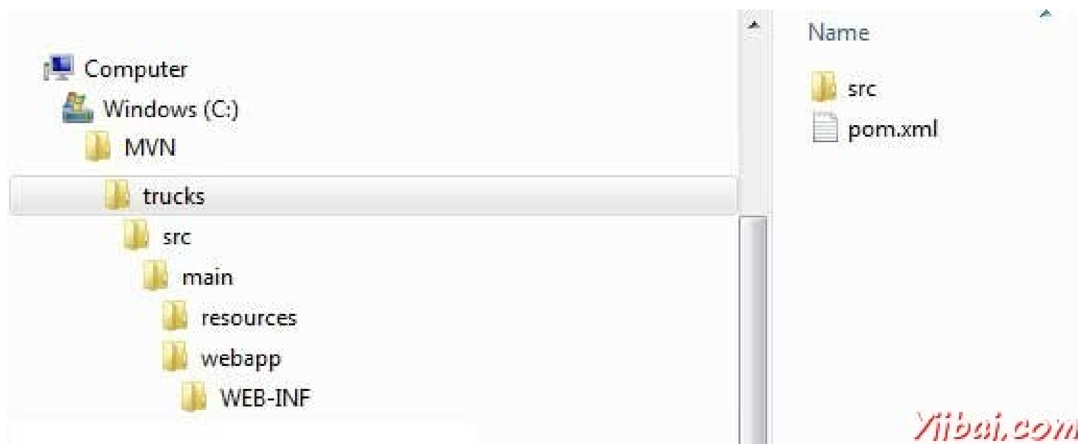
要创建一个简单的java web应用程序，我们将使用Maven的原型 - web应用插件。因此，让我们打开命令控制台，进入到C： MVN目录并执行以下命令mvn命令。

```
C:MVN>mvn archetype:generate
-DgroupId=com.companyname.automobile
-DartifactId=trucks
-DarchetypeArtifactId=maven-archetype-webapp
-DinteractiveMode=false
```

Maven会开始处理，并建立完整的基于Web的Java应用程序项目结构。

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] -------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [archetype:generate] (aggregator-style)
[INFO] -------------------------------------------------------------------
[INFO] Preparing archetype:generate
[INFO] No goals needed for project - skipping
[INFO] [archetype:generate {execution: default-cli}]
[INFO] Generating project in Batch mode
[INFO] -------------------------------------------------------------------
[INFO] Using following parameters for creating project
from Old (1.x) Archetype: maven-archetype-webapp:1.0
[INFO] -------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.companyname.automobile
[INFO] Parameter: packageName, Value: com.companyname.automobile
[INFO] Parameter: package, Value: com.companyname.automobile
[INFO] Parameter: artifactId, Value: trucks
[INFO] Parameter: basedir, Value: C:MVN
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:MVN    ru
[INFO] -------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -------------------------------------------------------------------
[INFO] Total time: 16 seconds
[INFO] Finished at: Tue Jul 17 11:00:00 IST 2012
[INFO] Final Memory: 20M/89M
[INFO] -------------------------------------------------------------------
```

现在去到C:/ MVN目录。您将看到创建了一个名为trucks （如artifactId指定）一个java应用程序项目。




Maven使用标准的目录结构。用上面的例子中，我们可以了解到以下关键概念

| 文件夹结构 | 描述 |
| --- | --- |
| trucks | contains src folder and pom.xml |
| src/main/webapp | contains index.jsp and WEB-INF folder. |
| src/main/webapp/WEB-INF | contains web.xml |
| src/main/resources | it contains images/properties files . |

# POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.companyname.automobile</groupId>
   <artifactId>trucks</artifactId>
   <packaging>war</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>trucks Maven Webapp</name>
   <url>http://maven.apache.org</url>
   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
   <build>
      <finalName>trucks</finalName>
   </build>
</project>
```

Maven还创建了一个示例JSP源文件

打开 **C: > MVN > trucks > src > main > webapp >** 文件夹，你会看到index.jsp。

```
<html>
   <body>
      <h2>Hello World!</h2>
   </body>
</html>
```

# 构建Web应用程序

让我们打开命令控制台，进入到C:MVN rucks目录并执行以下命令mvn命令。

```
C:MVN    rucks>mvn clean package
```

Maven将开始建设该项目。

```
[INFO] Scanning for projects...
[INFO] -------------------------------------------------------------------
[INFO] Building trucks Maven Webapp
[INFO]    task-segment: [clean, package]
[INFO] -------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (Cp1252 actually) to
copy filtered resources,i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] No sources to compile
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (Cp1252 actually) to
copy filtered resources,i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory
C:MVN    ruckssrc    est
esources
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] No sources to compile
[INFO] [surefire:test {execution: default-test}]
[INFO] No tests to run.
[INFO] [war:war {execution: default-war}]
[INFO] Packaging webapp
[INFO] Assembling webapp[trucks] in [C:MVN    rucks    arget    ruc
[INFO] Processing war project
[INFO] Copying webapp resources[C:MVN    ruckssrcmainwebapp]
[INFO] Webapp assembled in[77 msecs]
[INFO] Building war: C:MVN    rucks    arget    rucks.war
[INFO] -------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -------------------------------------------------------------------
[INFO] Total time: 3 seconds
[INFO] Finished at: Tue Jul 17 11:22:45 IST 2012
[INFO] Final Memory: 11M/85M
[INFO] -------------------------------------------------------------------
```

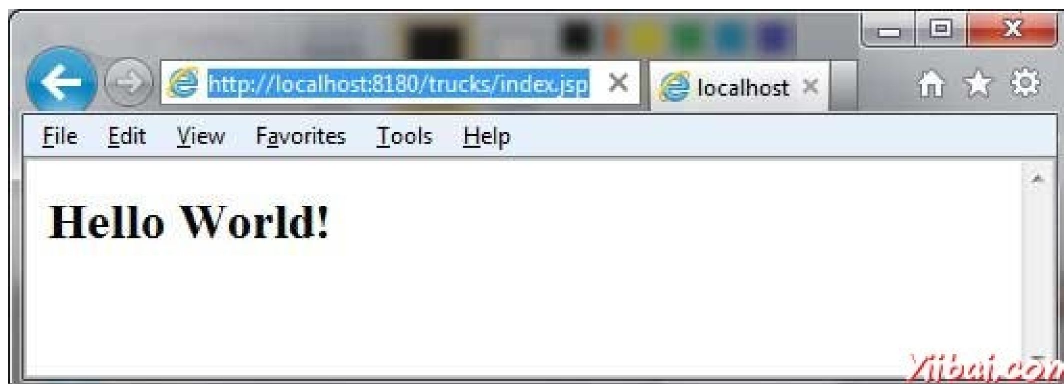
# 部署Web应用程序

现在复制创建的trucks.war 到**C: > MVN > trucks > target >**文件夹到web服务器的webapp目录下，然后重新启动Web服务器。

# 测试Web应用程序

使用URL运行Web应用程序：**http://<server-name>:<port-number>/trucks/index.jsp**

验证输出。



# Eclispe IDE集成Maven - Maven教程

Eclipse提供了一个很好的插件m2eclipse 无缝将Maven和Eclipse集成在一起。

m2eclipse一些特点如下

- 您可以从Eclipse运行Maven目标。
- 可以使用其自己的控制台查看Maven命令的输出在Eclipse里面。
- 你可以更新maven的依赖关系使用IDE。
- 您可以启动Maven在Eclipse中建立。
- 它的依赖管理基于Maven的pom.xml 在Eclipse构建路径。
- 它解决了从Eclipse工作区Maven的依赖关系，而不需要安装到本地Maven仓库（需要依赖项目在同一个工作区）。
- 它自动下载需要的依赖和源从远程Maven仓库。
- 它提供了向导，用于创建新的Maven项目，pom.xml和现有项目可让Maven支持
- 它提供了快速搜索远程Maven仓库的依赖

## 安装m2eclipse插件

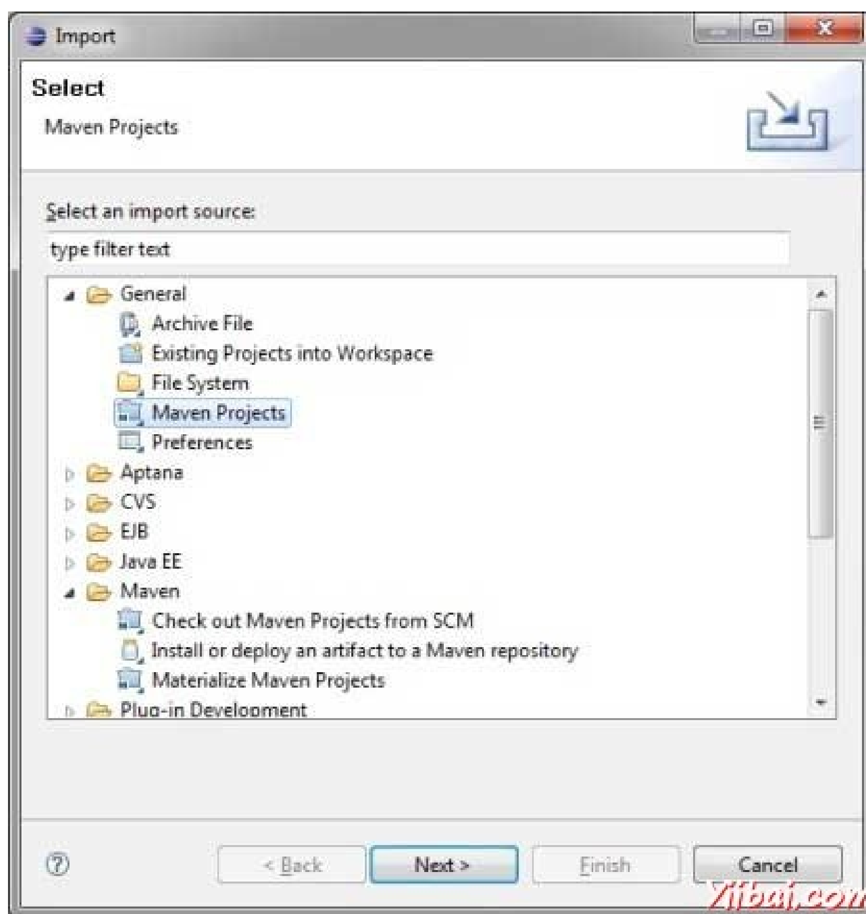
请使用以下链接之一安装m2eclipse：

| Eclipse | URL |
| --- | --- |
| Eclipse 3.5 (Gallileo) | Installing m2eclipse in Eclipse 3.5 (Gallileo) |
| Eclipse 3.6 (Helios) | Installing m2eclipse in Eclipse 3.6 (Helios) |

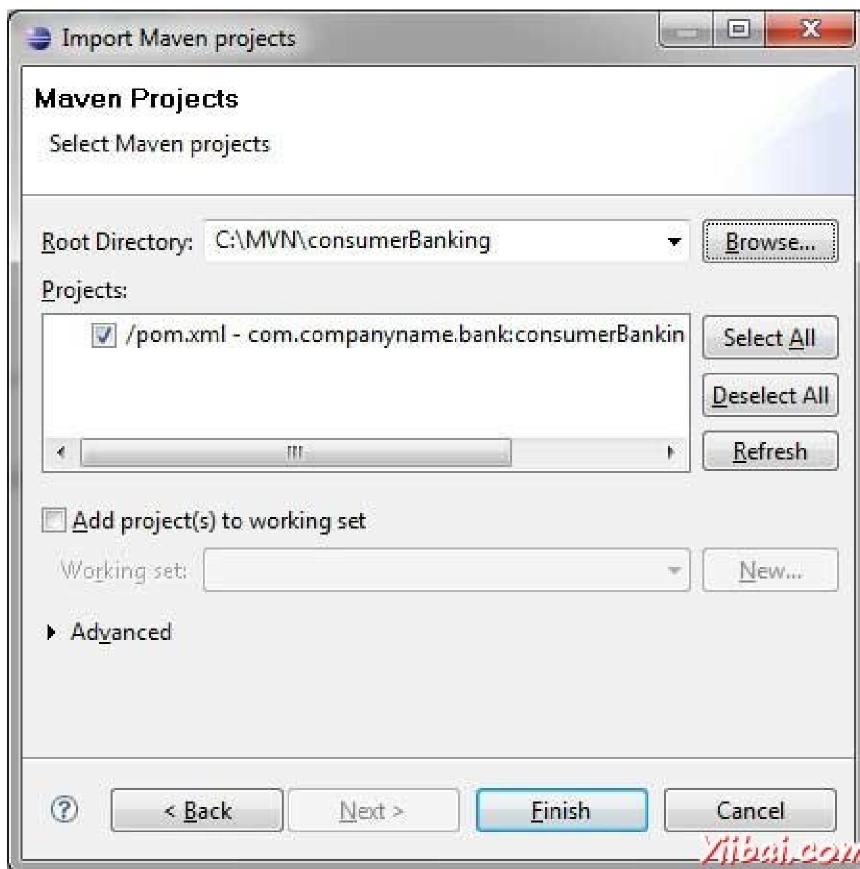
下面的例子将帮助您利用集成Eclipse和Maven。

## 导入Eclipse中Maven项目

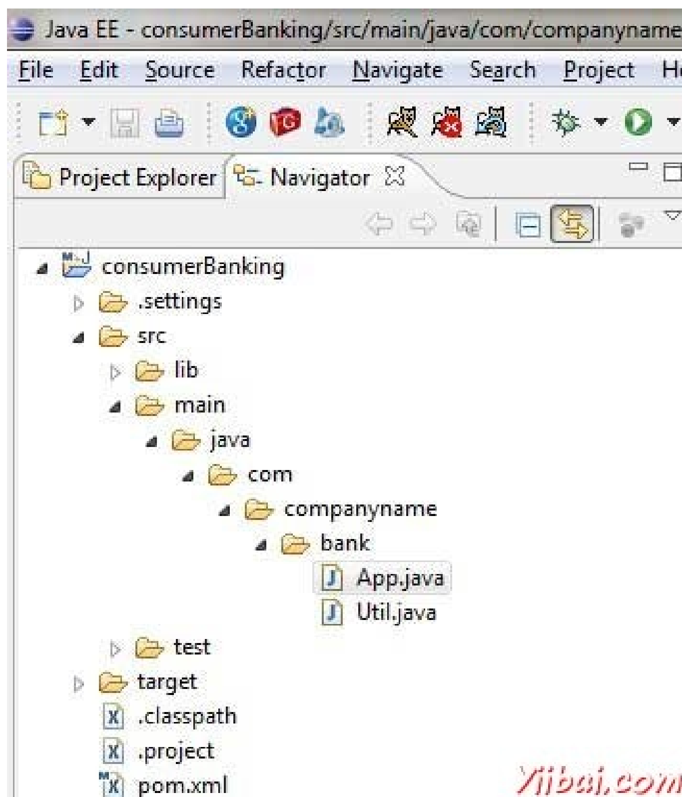
- 打开Eclipse.
- 选择File > Import > 选项.
- 选择Maven项目选项。单击Next按钮。



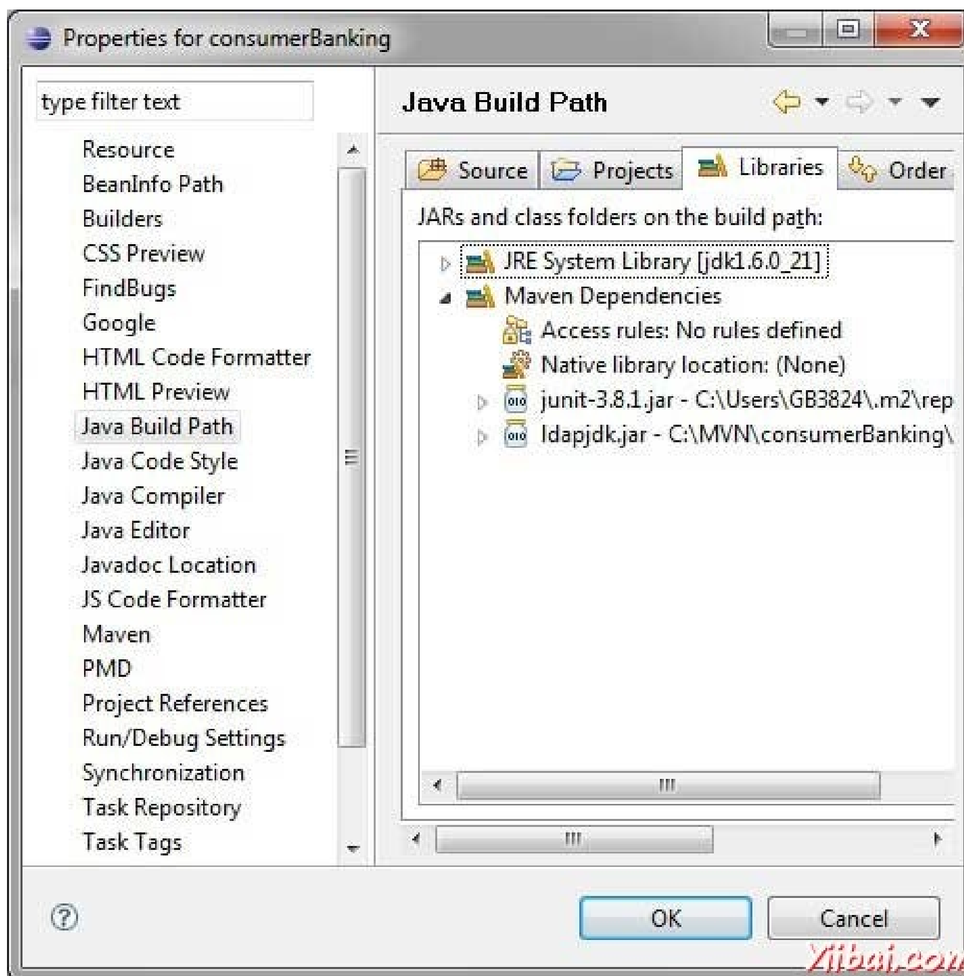

- 选择项目的位置，使用Maven创建一个项目。我们已经创建一个Java项目consumerBanking。看到Maven创建项目，看看如何创建使用Maven项目。
- 单击Finish按钮。




现在，你可以看到Maven项目在eclipse。




现在，看看consumerBanking项目属性。你可以看到，Eclipse已经添加Maven的依赖关系，以Java构建路径。




现在，它使用Eclipse的Maven来构建项目。

- 右键点击consumerBanking项目打开上下文菜单。
- 选择 Run 作为选项
- 然后maven的封装选项

Maven将开始建设该项目。你可以看到在Eclipse控制台输出

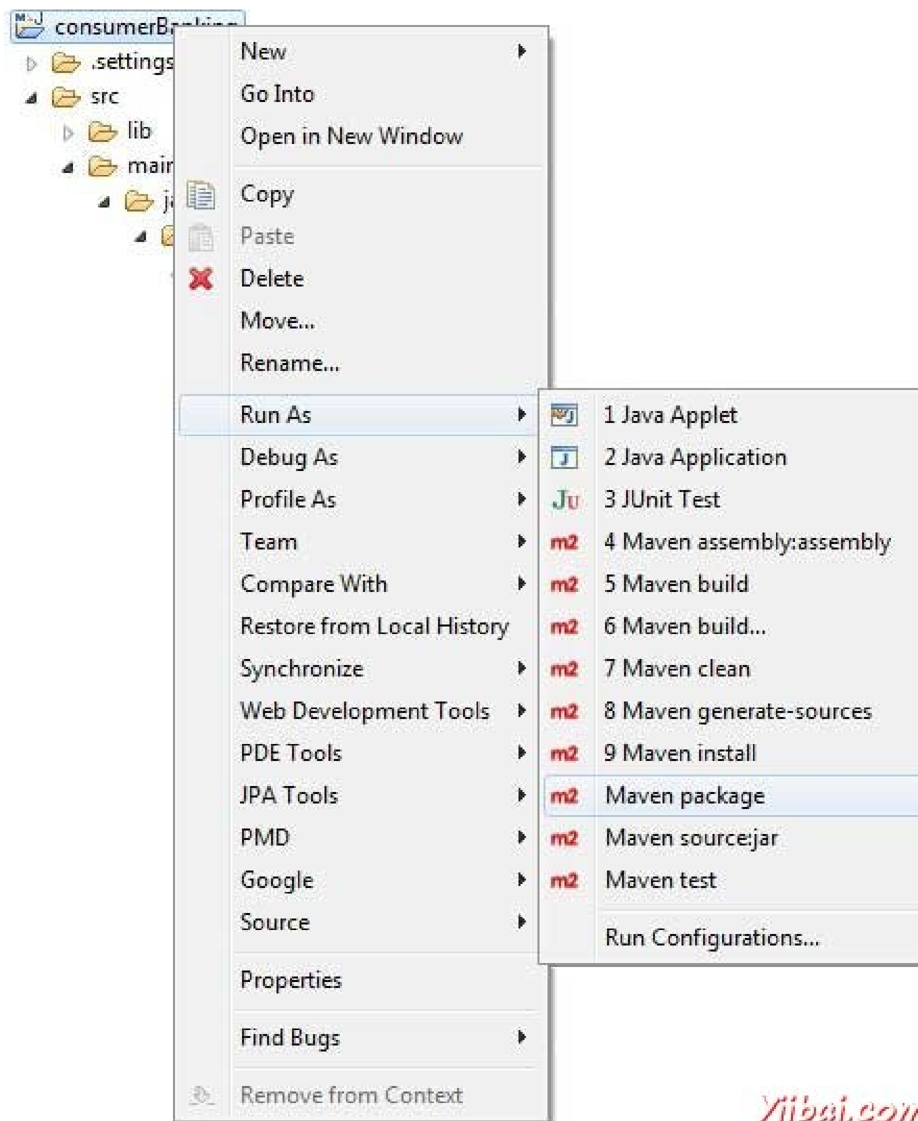
```
[INFO] Scanning for projects...
[INFO] -------------------------------------------------------------------
[INFO] Building consumerBanking
[INFO]
[INFO] Id: com.companyname.bank:consumerBanking:jar:1.0-SNAPSHOT
[INFO] task-segment: [package]
[INFO] -------------------------------------------------------------------
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory:
C:MVNconsumerBanking    argetsurefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.companyname.bank.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.0

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar]
[INFO] -------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -------------------------------------------------------------------
[INFO] Total time: 1 second
[INFO] Finished at: Thu Jul 12 18:18:24 IST 2012
[INFO] Final Memory: 2M/15M
[INFO] -------------------------------------------------------------------
```




现在，右键点击App.java。选择Run As选项。选择作为Java应用程序。

你会看到结果

# NetBeans IDE集成Maven - Maven教程

NetBeans6.7更新版本已经内置对Maven支持。如遇以前的版本，Maven插件在插件管理器中可用。我们正在使用NetBeans在这个例子中使用6.9。

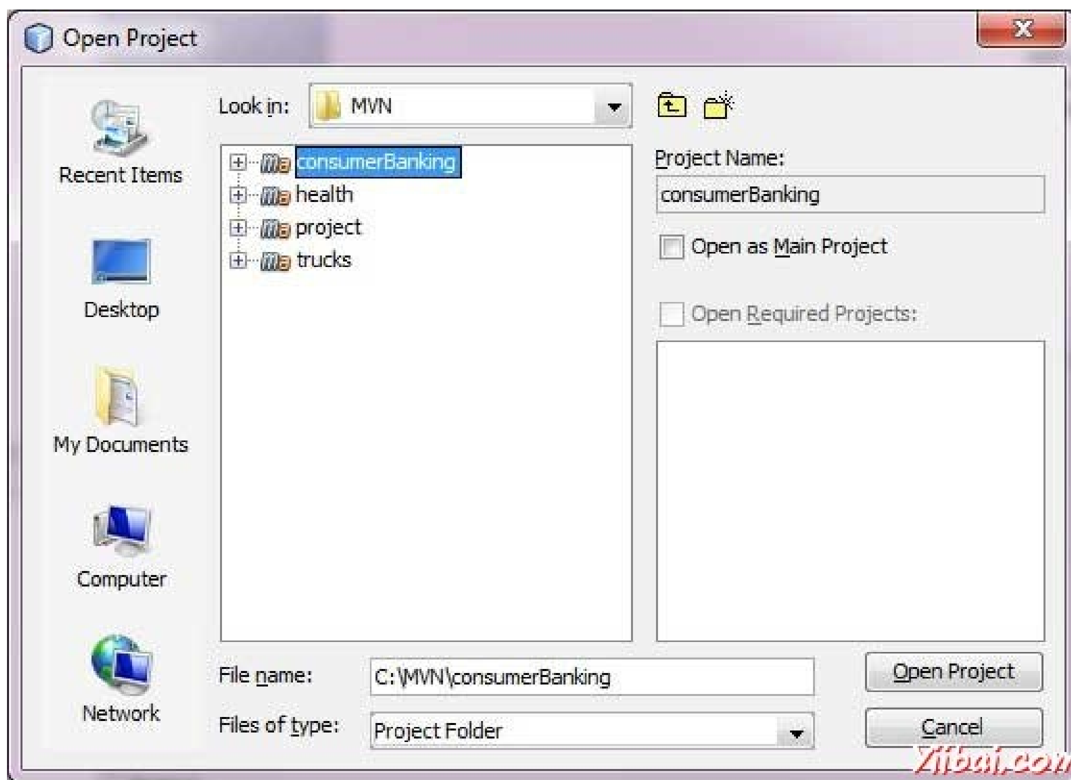
在NetBeans一些特点如下

- 您可以从NetBeans运行Maven目标。
- 您可以可以查看Maven命令的输出使用其自己的控制台在NetBeans里面。
- 你可以更新maven的依赖关系的IDE。
- 您可以启动Maven从内部的NetBeans版本。
- NetBeans不依赖自动管理基于Maven的pom.xml。
- NetBeans解决Maven的依赖关系从它的工作空间，而不需要安装到本地Maven仓库（需要依赖项目在同一个工作区）。
- NetBeans自动下载需要的依赖和源从远程Maven仓库。
- NetBeans提供向导，用于创建新的Maven项目及pom.xml
- NetBeans提供一个Maven资源库浏览器，使您可以查看您的本地存储库和注册的外部Maven仓库。

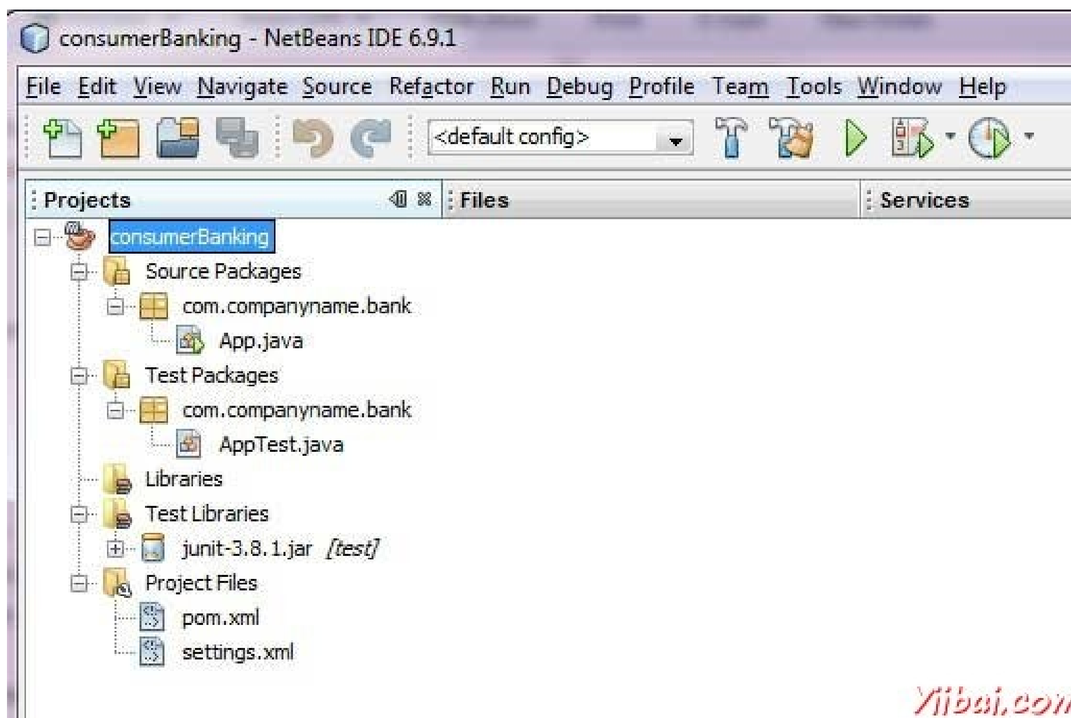
下面的例子将帮助您充分利用NetBeans的集成和Maven的好处。

## 打开NetBeans Maven项目

- 打NetBeans.
- 选择 File Menu > Open Project 选项.
- 选择项目的位置，使用Maven在那里创建了一个项目。我们已经创建一个Java项目consumerBanking。看到Maven创建项目，看看如何创建使用Maven项目。




现在，你可以看到Maven项目在NetBeans。看看consumerBanking项目库和测试库。可以看到，NetBeans的增加了Maven的依赖关系到它的构建路径。

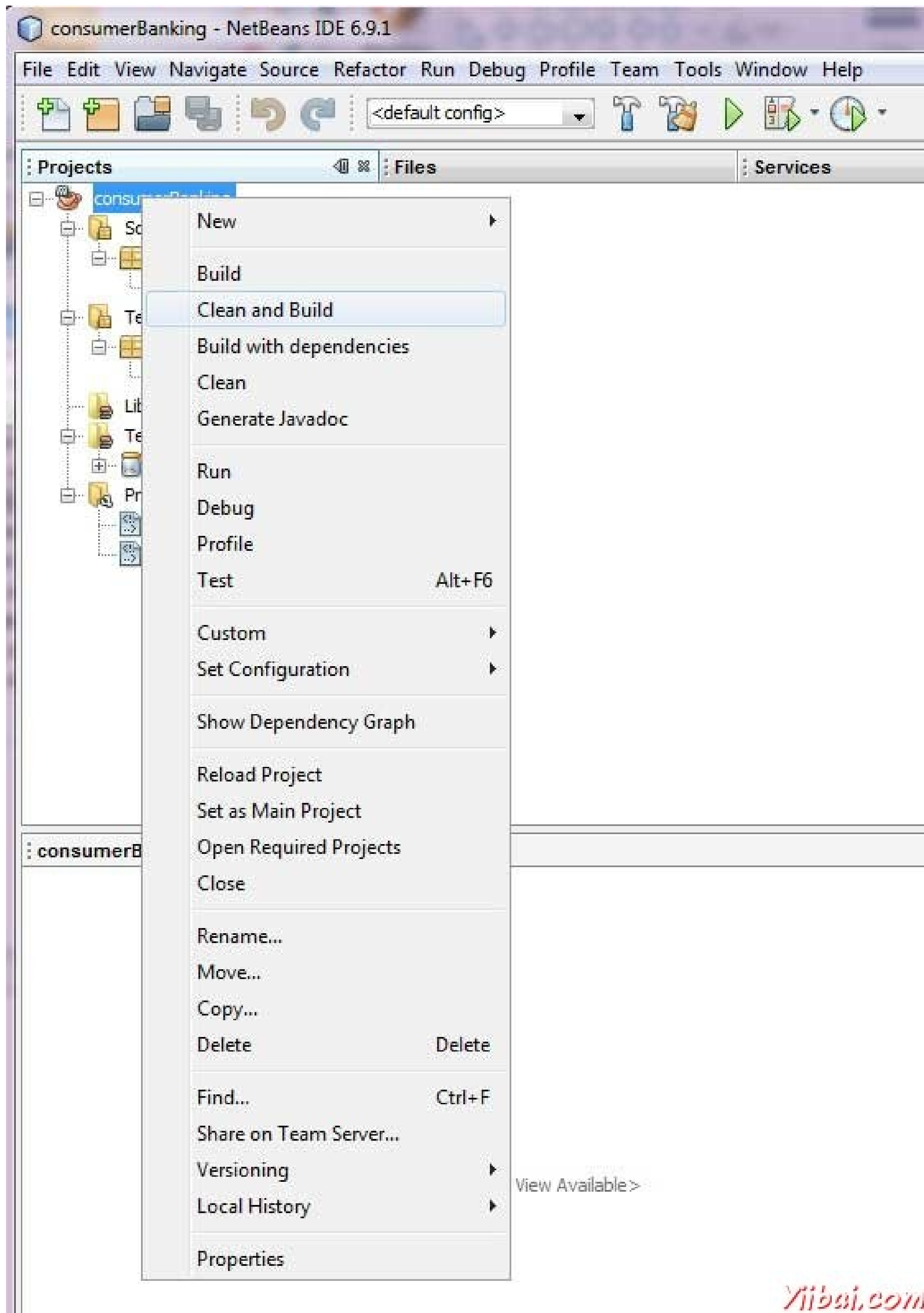



# 建立在NetBeans Maven项目

现在，它使用NetBeans的Maven来构建项目。

- 右键点击consumerBanking项目打开上下文菜单。

- 选择清理并生成可选项




Maven将开始建设该项目。你可以看到在NetBeans控制台输出

```
NetBeans: Executing 'mvn.bat -Dnetbeans.execution=true clean instal
NetBeans:      JAVA_HOME=C:Program FilesJavajdk1.6.0_21
Scanning for projects...
--------------------------------------------------------------------
Building consumerBanking
   task-segment: [clean, install]
--------------------------------------------------------------------
[clean:clean]
[resources:resources]
[WARNING] Using platform encoding (Cp1252 actually)
to copy filtered resources, i.e. build is platform dependent!
skip non existing resourceDirectory C:MVNconsumerBankingsrcmain
esources
[compiler:compile]
Compiling 2 source files to C:MVNconsumerBanking    argetclasses
[resources:testResources]
[WARNING] Using platform encoding (Cp1252 actually)
to copy filtered resources, i.e. build is platform dependent!
skip non existing resourceDirectory C:MVNconsumerBankingsrc    est
esources
[compiler:testCompile]
Compiling 1 source file to C:MVNconsumerBanking    arget    est-cla
[surefire:test]
Surefire report directory: C:MVNconsumerBanking    argetsurefire-re

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.companyname.bank.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.0

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[jar:jar]
Building jar: C:MVNconsumerBanking    argetconsumerBanking-1.0-SNAP
[install:install]
Installing C:MVNconsumerBanking    argetconsumerBanking-1.0-SNAPSHO
to C:UsersGB3824.m2
epositorycomcompanynameankconsumerBanking
1.0-SNAPSHOTconsumerBanking-1.0-SNAPSHOT.jar
--------------------------------------------------------------------
BUILD SUCCESSFUL
--------------------------------------------------------------------
Total time: 9 seconds
Finished at: Thu Jul 19 12:57:28 IST 2012
Final Memory: 16M/85M
--------------------------------------------------------------------
```

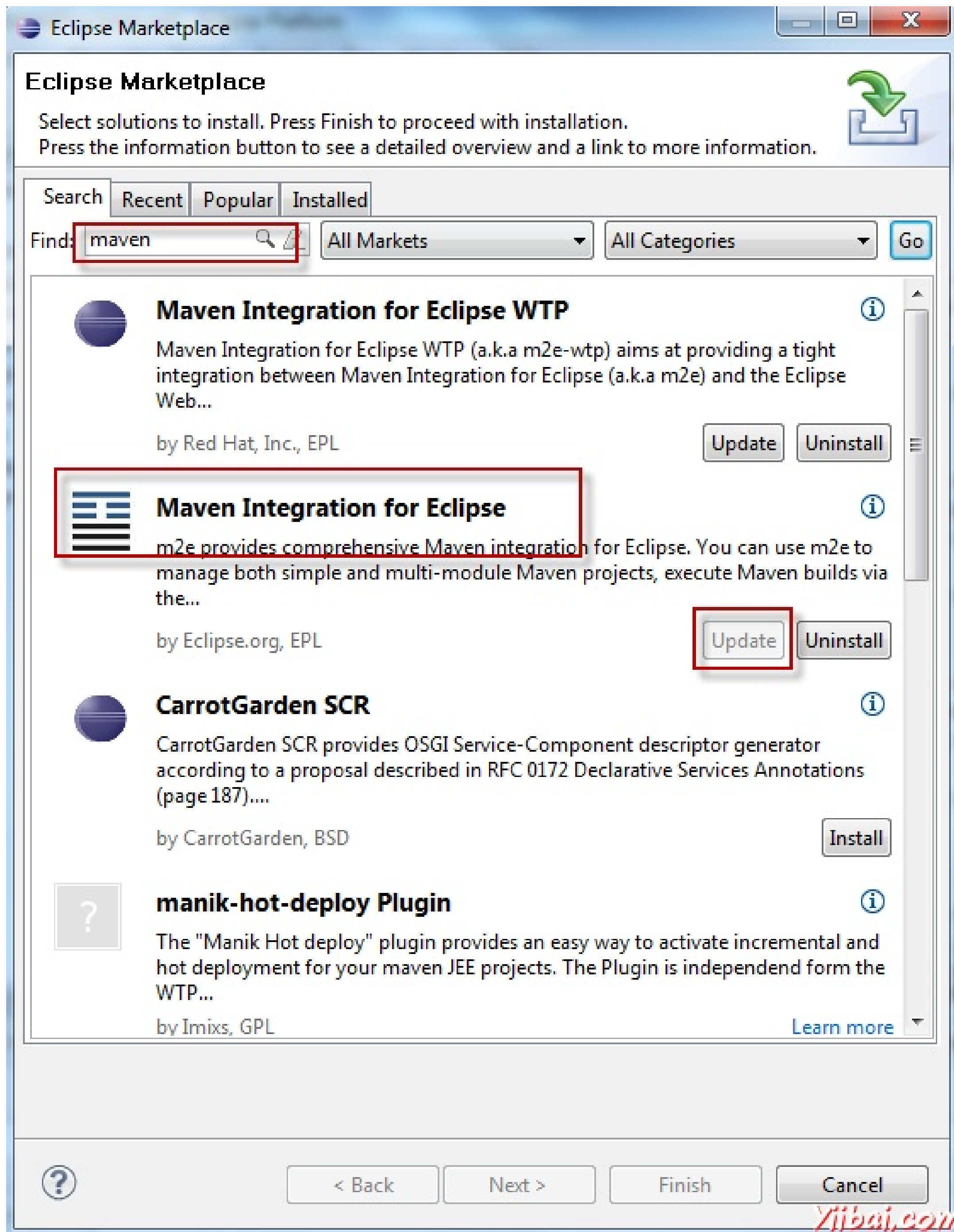
# 在NetBeans中运行的应用程序

现在，右键点击App.java。选择Run档选项。你会看到在NetBeans控制台的结果。

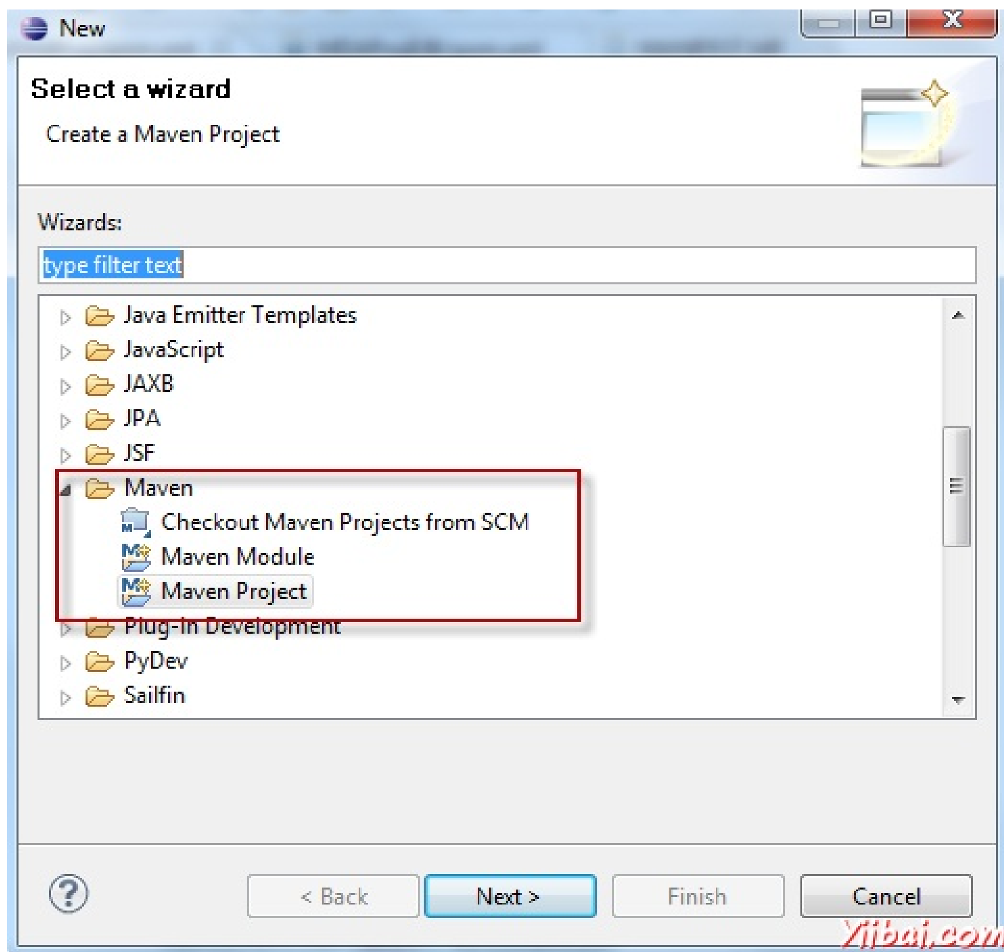
```
NetBeans: Executing 'mvn.bat -Dexec.classpathScope=runtime
-Dexec.args=-classpath %classpath com.companyname.bank.App
-Dexec.executable=C:Program FilesJavajdk1.6.0_21injava.exe
-Dnetbeans.execution=true process-classes
org.codehaus.mojo:exec-maven-plugin:1.1.1:exec'
NetBeans:      JAVA_HOME=C:Program FilesJavajdk1.6.0_21
Scanning for projects...
---------------------------------------------------------------
Building consumerBanking
   task-segment: [process-classes,
   org.codehaus.mojo:exec-maven-plugin:1.1.1:exec]
---------------------------------------------------------------
[resources:resources]
[WARNING] Using platform encoding (Cp1252 actually)
to copy filtered resources, i.e. build is platform dependent!
skip non existing resourceDirectory C:MVNconsumerBankingsrcmain
esources
[compiler:compile]
Nothing to compile - all classes are up to date
[exec:exec]
**Hello World!**
---------------------------------------------------------------
BUILD SUCCESSFUL
---------------------------------------------------------------
Total time: 1 second
Finished at: Thu Jul 19 14:18:13 IST 2012
Final Memory: 7M/64M
---------------------------------------------------------------
```

# Eclipse构建Maven项目 - Maven教程

**1.** 安装**m2eclipse**插件 要用Eclipse构建Maven项目，我们需要先安装meeclipse插件 点击eclipse菜单栏Help->Eclipse Marketplace搜索到插件Maven Integration for Eclipse 并点击安装即可，如下图：




安装成成之后我们在Eclipse菜单栏中点击File->New->Other,在弹出的对话框中会看到如下图所示：

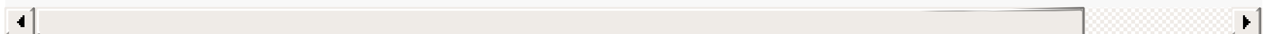


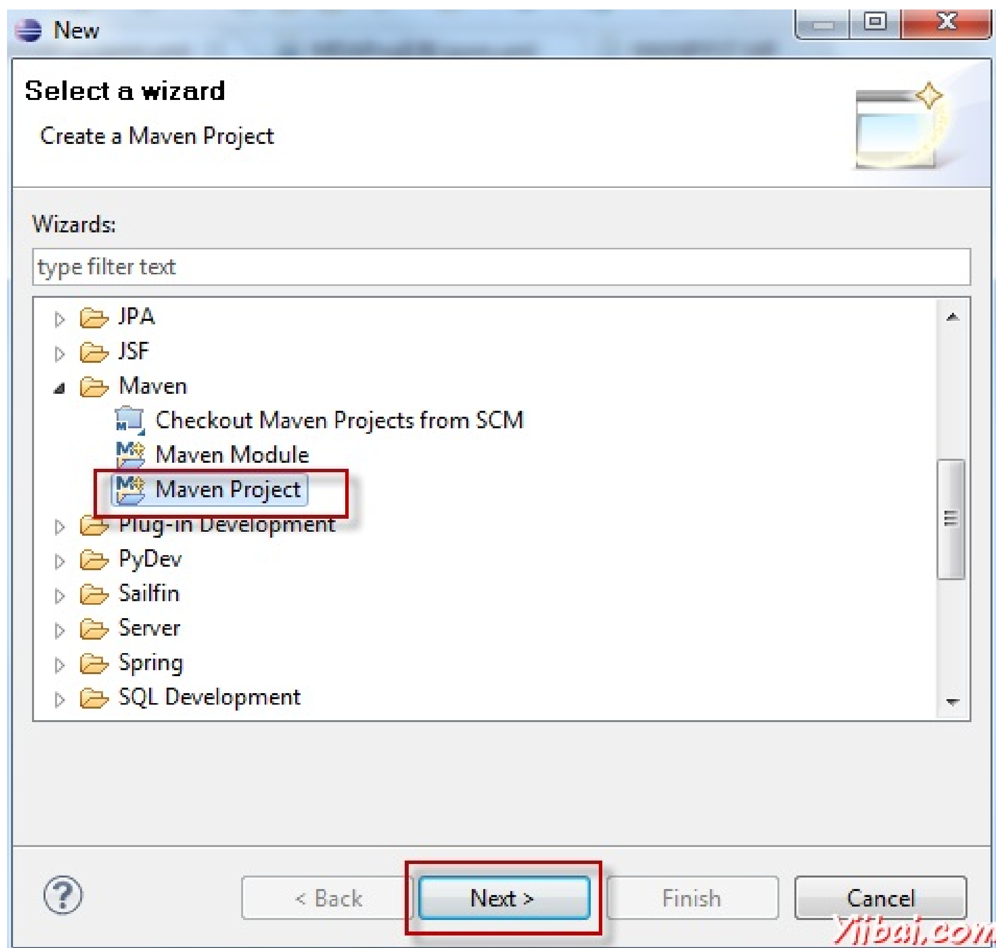

## 2. 构建Maven项目

```
以eclipse3.6为例

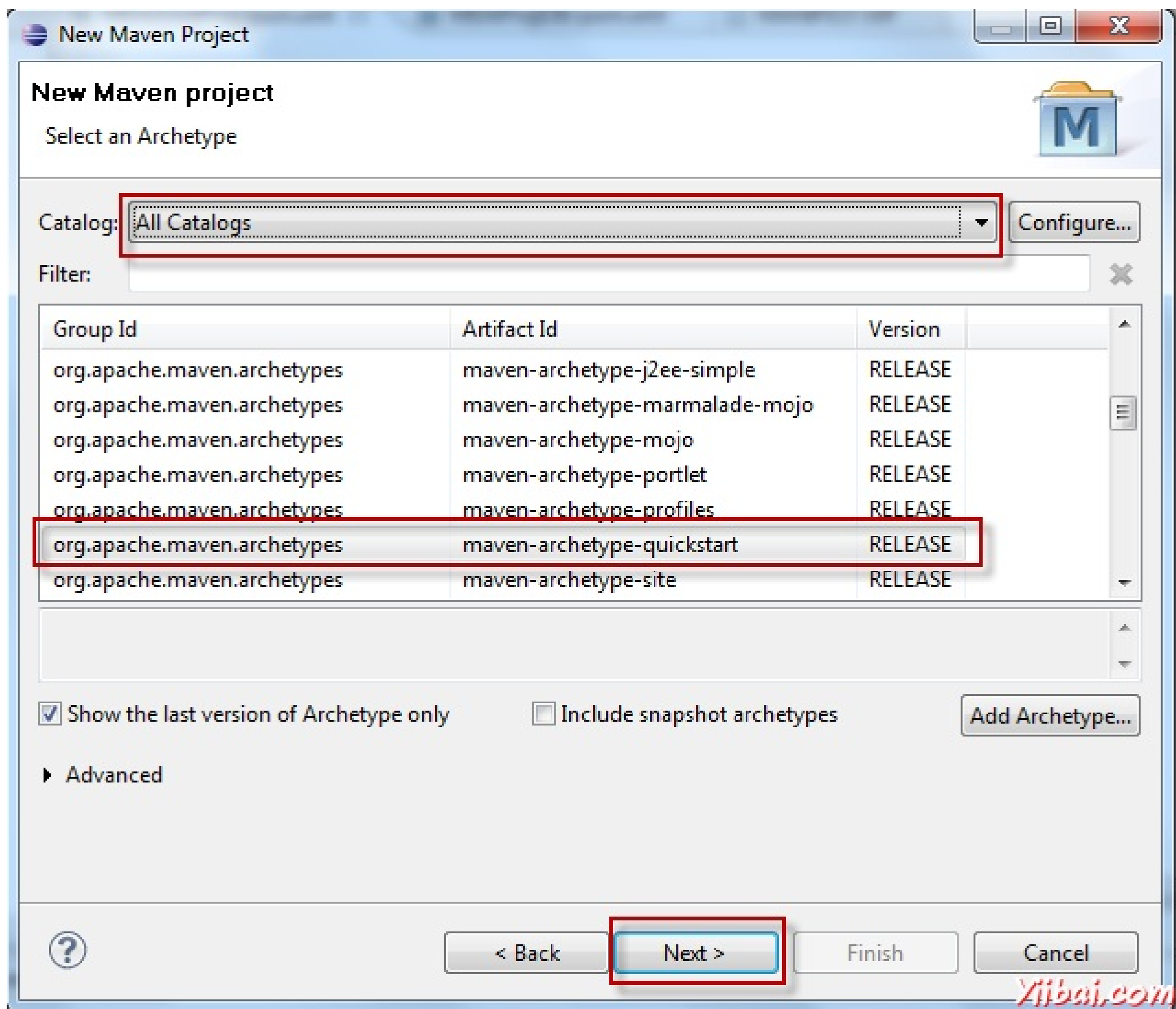
1） 创建简单Maven项目

    点击Eclipse菜单栏File-&gt;New-&gt;Ohter-&gt;Maven得到如下图所示：
```

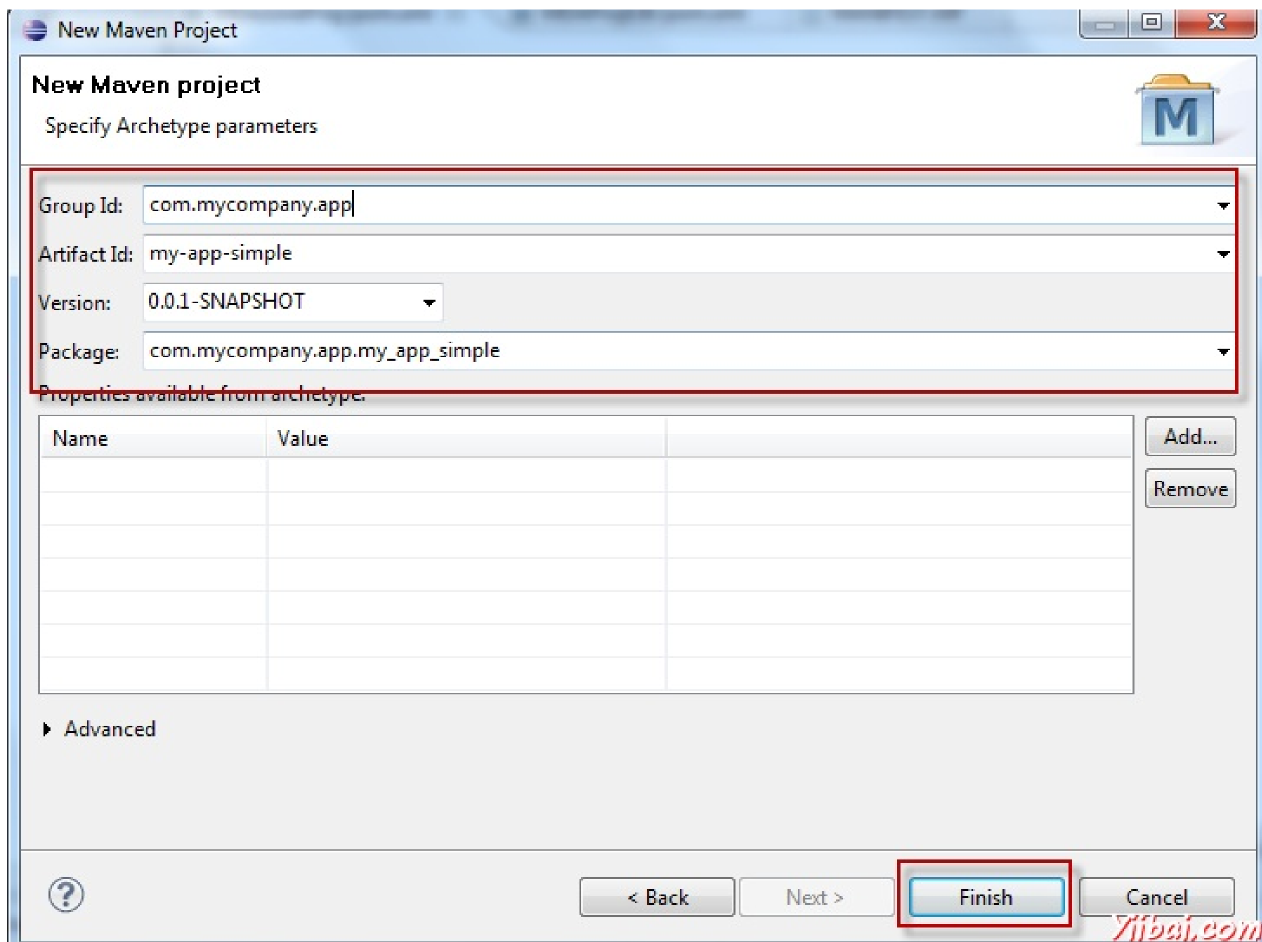





选中Maven Project并点击Next，到下一个对话框继续点击Next得到如下对话框

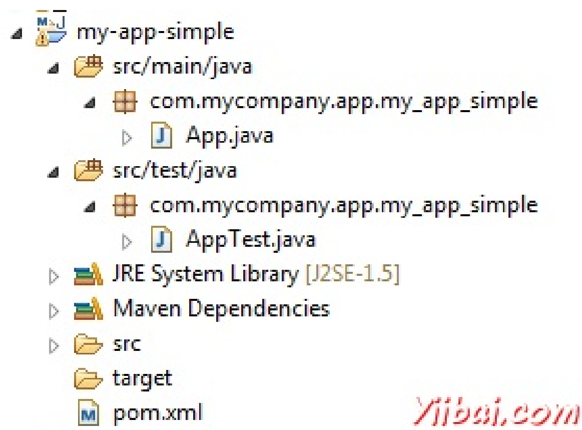



如图示操作，选择maven-archetype-quickstart，点击Next




按图示填写好groupId, artfactId,version等信息， 点击Finish。

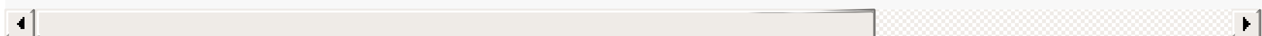
由此我们成功创建了一个简单的Maven项目，项目结构如图所示




2) 创建Maven web项目

操作跟创建简单Maven项目类似，点击Eclipse菜单File->New->Other->

在选择maven-archetype的界面进行如下操作：



New Maven Project

New Maven project

Select an Archetype

Catalog: Internal Configure...

Filter: webapp

| Group Id | Artifact Id | Version |
|---|---|---|
| org.apache.cocoon | cocoon-22-archetype-webapp | RELEASE |
| org.apache.maven.archetypes | maven-archetype-webapp | RELEASE |

A simple Java web application

Show the last version of Archetype only Include snapshot archetypes Add Archetype...

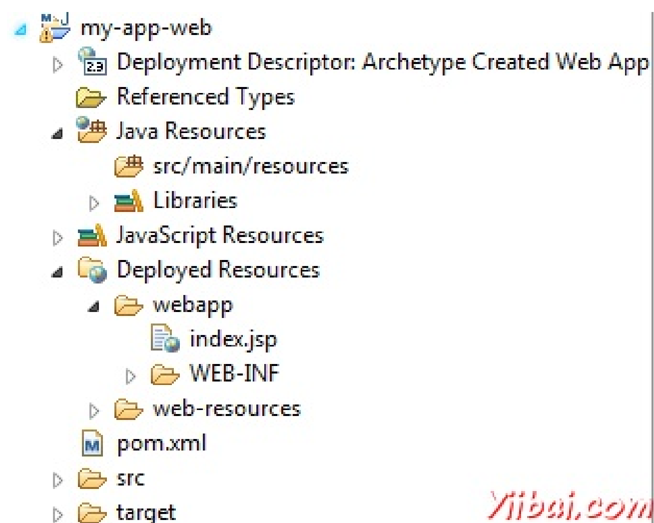
Advanced

< Back Next > Finish Cancel

Yiibai.com

点击Next,填写好相应的groupId,artifactId,version等信息， 点击Finish

得到的Maven web项目结构如下图所示：




右击项目， 点击Properties->Project Facets

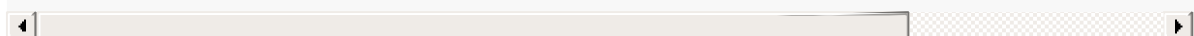
如上图可以看到项目为web2.3 java1.5 当然我们也可以改成我们所需要的版本，打开xml文件my-app-web/.settings/org.eclipse.wst.common.project.facet.core.xml，进行修改即可：

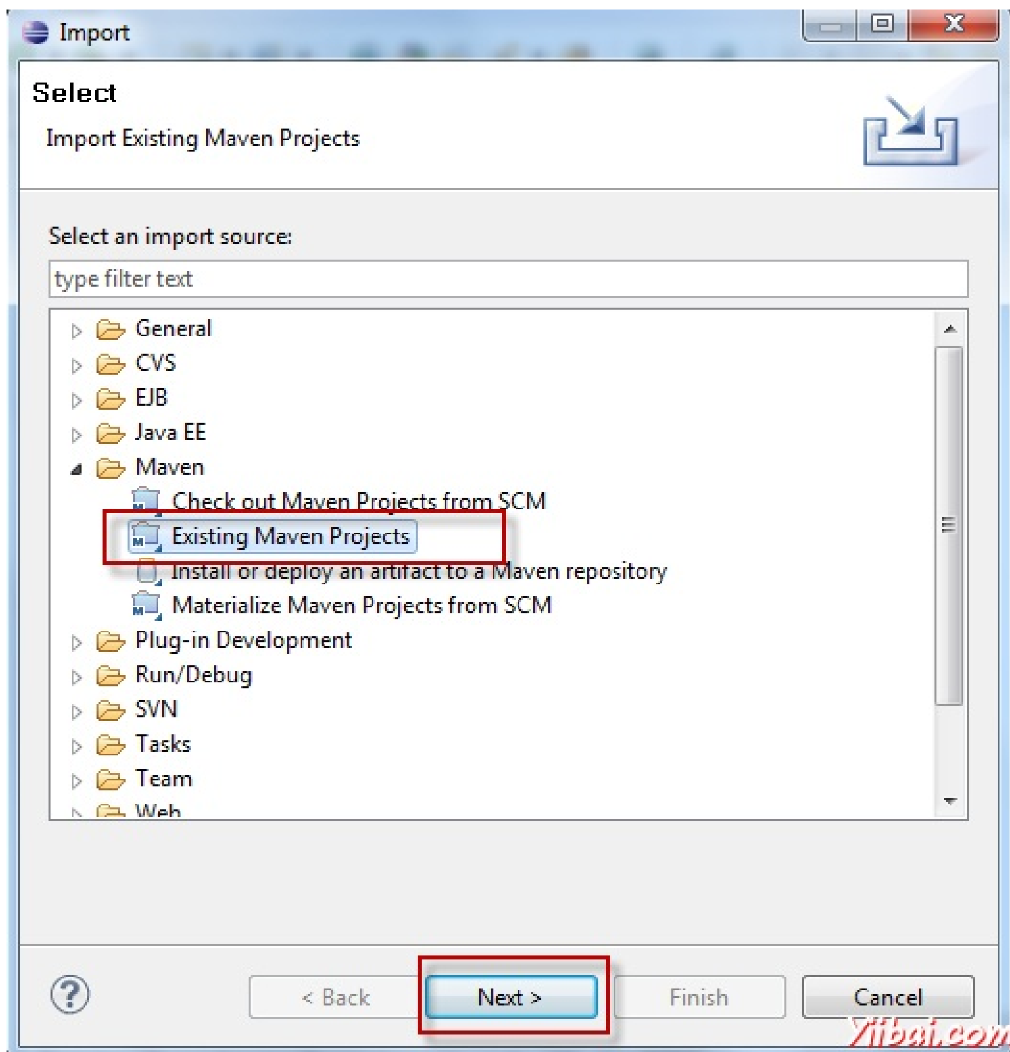
Xml代码 ☆

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <faceted-project>
3. <fixed facet="wst.jsdt.web"/>
4. <installed facet="java" version="1.5"/>
5. <installed facet="jst.web" version="2.3"/>
6. <installed facet="wst.jsdt.web" version="1.0"/>
7. </faceted-project>
```

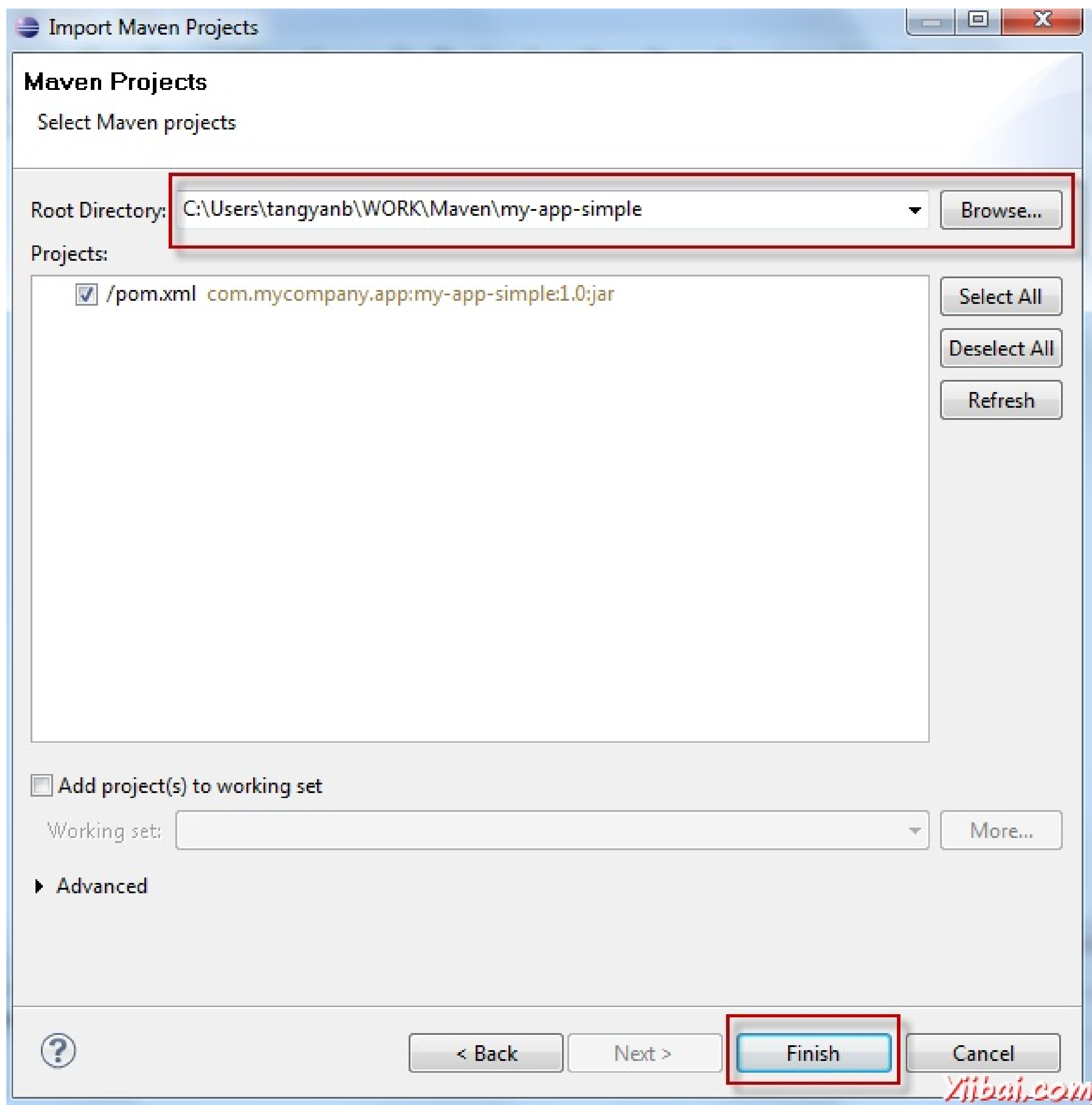
3）导入Maven项目

```
在Eclipse project explorer中右击，在弹出框中选择import，得到如下图所
```






选择Existing Maven Projects，并点击Next，得到如下图所示对话框：

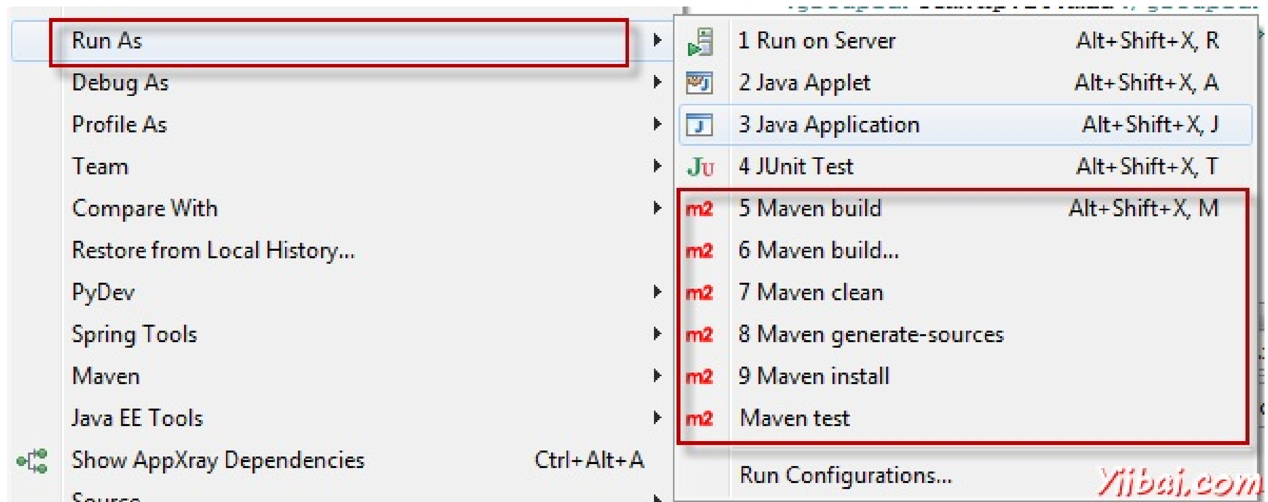



选择一个已经创建好的Maven项目，并点击Finish。

由此，导入Maven项目成功

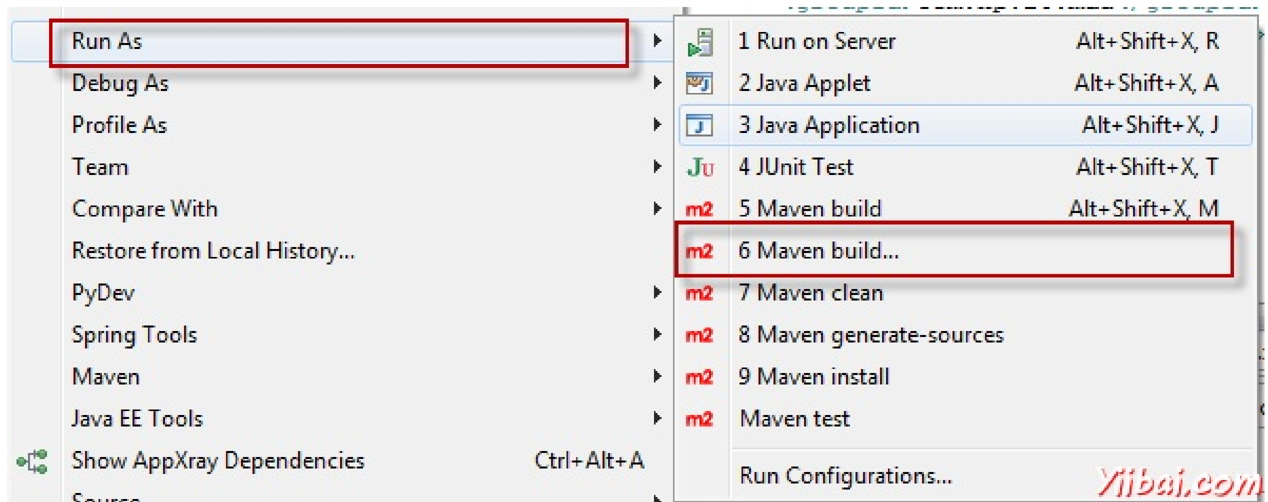
### 3. 运行Maven命令

```
右击项目，点击Run as，如下图：
```




即可看到有很多现有的maven命令，点击即可运行，并在控制台可以看到运行信息

如果你想运行的maven命令在这里没有找到，点击Maven build创建新的命令，操作如下图所示：




如下图填入Maven命令，点击Run即可

Edit Configuration

Edit configuration and launch.

Name: my-app-web (2)

Main | JRE | Refresh | Environment | Common

Base directory:

C:/Users/tangyanb/WORK/Maven/my-app-web

Browse Workspace... | Browse File System... | Variables...

Goals: clean package | Select...

Profiles:

Offline | Update Snapshots

Debug Output | Skip Tests | Non-recursive

Resolve Workspace artifacts

| Parameter Name | Value |
| --- | --- |

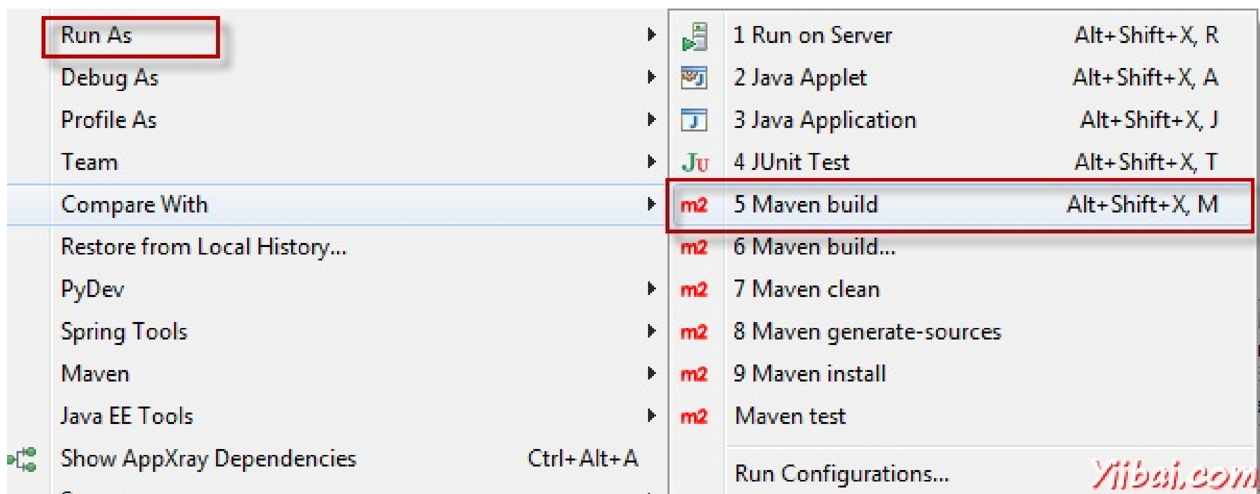
Add... | Edit... | Remove

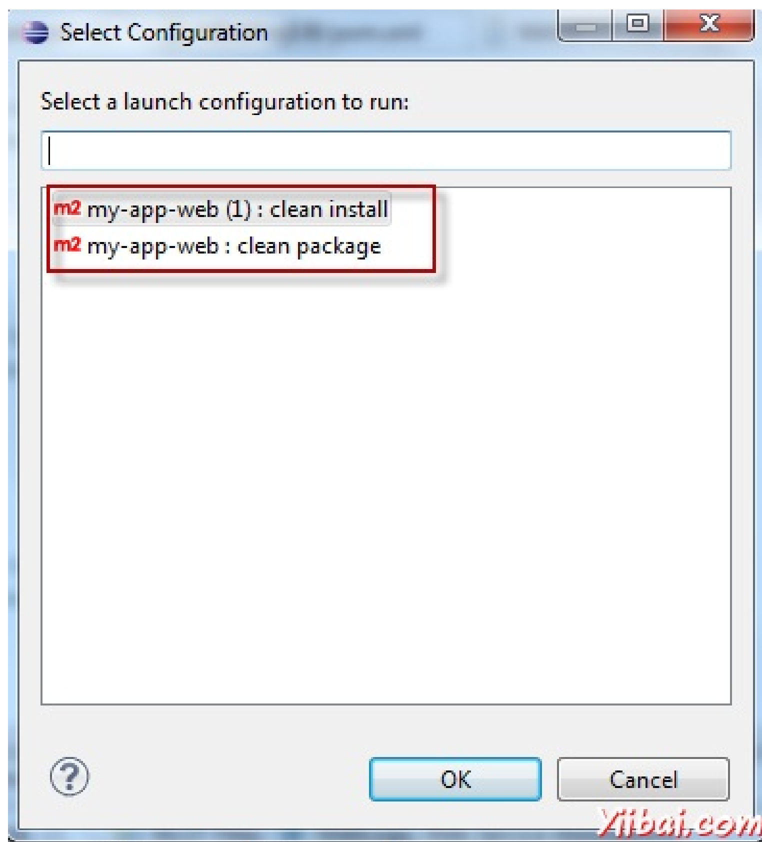
Maven Runtime: Embedded (3.0.2/1.0.200.20111228-1245) | Configure...

Apply | Revert

Run | Close

新增的maven命令可以通过如下方式找到，并再次运行：




Select Configuration
Select a launch configuration to run:
m2 my-app-web (1) : clean install
m2 my-app-web : clean package
OK
Cancel
Yiibai.com

# 转换基于Maven的Web应用程序支持Eclipse IDE - Maven教程

在上一节教程中，使用Maven创建了一个Web应用程序。这里有一个指南，告诉你如何转换Web应用程序到Eclipse IDE支持的形式。

注意，通过WTP工具Eclipse IDE支持Web应用程序，所以需要让基于Maven的项目支持它。

## 1. mvn eclipse:eclipse -Dwtpversion=2.0

要转换一个基于Maven的Java项目支持IDE，使用此命令：

```
mvn eclipse:eclipse
```

对于Web应用程序，需要额外的参数，使其支持 Eclipse WTP，应该使用这个命令：

```
mvn eclipse:eclipse -Dwtpversion=2.0
```

看看其输出 ...

```
_C:\worksp> mvn archetype:generate -DgroupId=com.yiibai -DartifactI
C:\worksp\yiibaiweb-core>mvn eclipse:eclipse -Dwtpversion=2.0
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------
[INFO] Building yiibaiweb-core 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.10:eclipse (default-cli) > genera
@ yiibaiweb-core >>>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.10:eclipse (default-cli) < genera
@ yiibaiweb-core <<<
[INFO]
[INFO] --- maven-eclipse-plugin:2.10:eclipse (default-cli) @ yiibai
[INFO] Adding support for WTP version 2.0.
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launchin
ER
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "yiibaiweb-core" to C:\worksp\yiib
[INFO]
[INFO] ------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
[INFO] Total time: 2.038 s
[INFO] Finished at: 2015-11-02T20:30:36+08:00
[INFO] Final Memory: 13M/114M
[INFO] ------------------------------------------------------------------
```

## 2. Eclipse WTP

标准Eclipse的".classpath"和".project"文件被创建。你会发现创建一个新的".setting" 文件夹，里面包含两个文件"org.eclipse.wst.common.component"和"org.eclipse.wst.common.project.facet.core.xml" 都是WTP或Faces文件用来支持Eclipse。

*File : org.eclipse.wst.common.project.facet.core.xml*

```
<faceted-project><fixed facet="jst.java"/><fixed facet="jst.web"/><
```

注意 使用JDK1.4 和 Maven2.X生成的 Web应用程序(见上文)，这是相当过时，需要将其升级到最新的JDK版本。

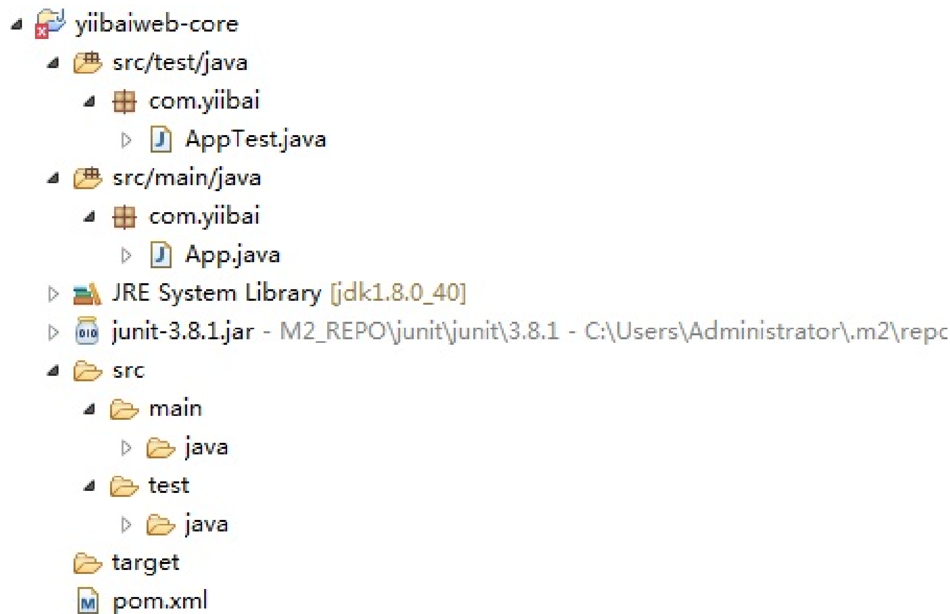
*File : org.eclipse.wst.common.component*

```
<project-modules id="moduleCoreId"project-version="2.0"><wb-module
```

# 3. 导入到Eclipse IDE

现在，我们已经拥有了一个 Eclipse 的 Web应用程序需要配置和文件，那么，就可以开始导入基于Maven构建Web应用程序到Eclipse IDE中去了。

步骤：在Eclipse IDE 的菜单栏，File -> Import... -> General -> Existing Projects into Workspace -> 选择根目录(选择项目文件夹中)-> 完成。




完事大吉！

# 参考

1. http://maven.apache.org/plugins/maven-eclipse-plugin/eclipse-mojo.html
2. http://maven.apache.org/plugins/maven-eclipse-plugin/wtp.html
3. http://wiki.eclipse.org/WTP_FAQ#What_version_of_Eclipse_does_WTP_work_with.3F
4. Unsupported WTP version: 1.5. This plugin currently supports only the following versions: 1.0 R7

# 使用Maven模板创建项目 - Maven教程

在本教程中，我们将向你展示如何使用mvn archetype:generate从现有的Maven模板列表中生成项目。在Maven 3.3.3，有超过1000+个模板，Maven 团队已经过滤掉一些无用的模板。

通常情况下，我们只需要使用下面的两个模板：

1. maven-archetype-webapp – Java Web Project (WAR)
2. maven-archetype-quickstart – Java Project (JAR)

## 1. Maven 1000+ 模板

如果键入命令mvn archetype:generate，1000 +模板会被提示在屏幕上，你没有办法看到它，或者选择什么。为了解决这个问题，输出模板列表，像这样保存为文本文件：

```
C:\worksp> mvn archetype:generate > templates.txt //waiting few sec
Press CTRL + C

C:\worksp> vim templates.txt
```

## 2. Maven archetype:generate

步骤来指导你如何从现有 Spring-Hibernate 模板来构建Web项目：

2.1 列出 Maven 的模板：

```
C:\worksp> mvn archetype:generate
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > gene
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < gene
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ star
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.

Choose archetype:
1: remote -> am.ik.archetype:maven-reactjs-blank-archetype (Blank P
2: remote -> am.ik.archetype:msgpack-rpc-jersey-blank-archetype (Bl
3: remote -> am.ik.archetype:mvc-1.0-blank-archetype (MVC 1.0 Blank
4: remote -> am.ik.archetype:spring-boot-blank-archetype (Blank Pro
5: remote -> am.ik.archetype:spring-boot-docker-blank-archetype (Do
6: remote -> am.ik.archetype:spring-boot-gae-blank-archetype (GAE B
7: remote -> am.ik.archetype:spring-boot-jersey-blank-archetype (Bl
8: remote -> at.chrl.archetypes:chrl-spring-sample (Archetype for S
```

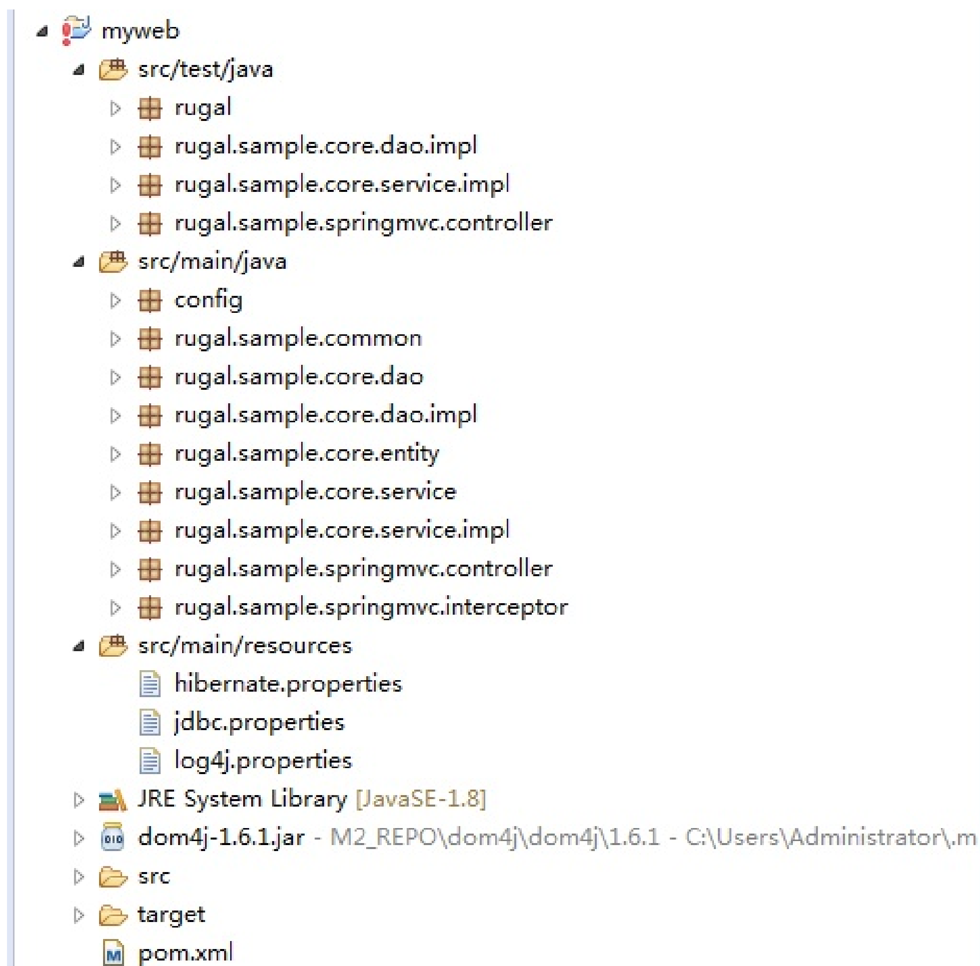
2.2 选择数字 “314” 来使用 ml.rugal.archetype:springmvc-spring-hibernate 模板，并填写详细信息：

注意，这个数字314可能在您的环境有所不同。寻找正确的数字应该看看在上面的步骤1中列出的技术。

```
1445: remote -> us.fatehi:schemacrawler-archetype-plugin-lint (-)
Choose a number or apply filter (format: [groupId:]artifactId, case
ntains): 674: 477
Choose ml.rugal.archetype:springmvc-spring-hibernate version:
1: 0.1
2: 0.2
3: 0.3
4: 0.4
5: 0.5
6: 0.6
Choose a number: 6:
Downloading: https://repo.maven.apache.org/maven2/ml/rugal/archetyp
pring-hibernate/0.6/springmvc-spring-hibernate-0.6.jar
Downloaded: https://repo.maven.apache.org/maven2/ml/rugal/archetype
ring-hibernate/0.6/springmvc-spring-hibernate-0.6.jar (30 KB at 6.8
Downloading: https://repo.maven.apache.org/maven2/ml/rugal/archetyp
pring-hibernate/0.6/springmvc-spring-hibernate-0.6.pom
Downloaded: https://repo.maven.apache.org/maven2/ml/rugal/archetype
ring-hibernate/0.6/springmvc-spring-hibernate-0.6.pom (4 KB at 5.3
```

```
Define value for property 'groupId': : com.yiibai.web
Define value for property 'artifactId': : myweb
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  com.yiibai.web: :
Confirm properties configuration:
groupId: com.yiibai.web
artifactId: myweb
version: 1.0-SNAPSHOT
package: com.yiibai.web
 Y: : y
[INFO] ------------------------------------------------------------
---
[INFO] Using following parameters for creating project from Archety
-spring-hibernate:0.6
[INFO] ------------------------------------------------------------
---
[INFO] Parameter: groupId, Value: com.yiibai.web
[INFO] Parameter: artifactId, Value: myweb
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.yiibai.web
[INFO] Parameter: packageInPathFormat, Value: com/yiibai/web
[INFO] Parameter: package, Value: com.yiibai.web
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: groupId, Value: com.yiibai.web
[INFO] Parameter: artifactId, Value: myweb
[INFO] project created from Archetype in dir: C:\worksp\myweb
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 05:07 min
[INFO] Finished at: 2015-11-03T07:10:56+08:00
[INFO] Final Memory: 16M/176M
[INFO] ------------------------------------------------------------
```

它会生成以下项目文件夹和文件。






图片: 生成Eclipse IDE中的项目结构。

注意，要导入项目到Eclipse中，键入命令mvn eclipse:eclipse，并导入它作为一个正常的项目：

```
c:\worksp> cd myweb
c:\worksp>myweb> mvn eclipse:eclipse
```

# 3. 更多示例

如果您知道哪个 archetypeArtifactId 使用，只需跳过交互模式命令：

3.1 maven-archetype-quickstart (Java Project)

```
$ mvn archetype:generate -DgroupId=com.yiibai.core -DartifactId=Pr
```

3.2 maven-archetype-webapp (Java Web Project)

```
$ mvn archetype:generate -DgroupId=com.yiibai.web -DartifactId=Proj
```

## 参考

1. Newbie: maven archetype:generate
2. Guide to Creating Archetypes
3. Maven Archetype Plugin – Usage
4. personal site for Rugal Bernstein

# 使用Maven构建项目 - Maven教程

要构建一个基于Maven的项目，打开控制台，进入到 pom.xml 文件所放的项目文件夹，并发出以下命令：

```
mvn package
```

这将执行Maven的“package”阶段。

**Maven**构建生命周期 Maven是分阶段运行，阅读 默认的Maven构建生命周期文章。因此，执行“package”阶段的时候，所有阶段 – “validate“, “compile” 和 “test“, 包括目前的阶段“package”将被执行。

## “mvn package” 示例

当你运行“mvn package”命令，它会编译源代码，运行单元测试和包装这取决于在pom.xml文件的“packaging”标签。 例如，

1. If “packaging” = jar, 将您的项目打包成一个“jar”文件，并把它变成你的目标文件夹。

*File : pom.xml*

```
<project...><modelVersion>4.0.0</modelVersion><groupId>com.yiibai</
```

2. 如果 “packaging” = war,将您的项目打包成“war”文件，并把它变成目标文件夹。

*File : pom.xml*

```
<project...><modelVersion>4.0.0</modelVersion><groupId>com.yiibai</
```

# 使用Maven清理项目 - Maven教程

在基于Maven的项目中，很多缓存输出在“target”文件夹中。如果想建立项目部署，必须确保清理所有缓存的输出，从面能够随时获得最新的部署。

要清理项目缓存的输出，发出以下命令：

```
mvn clean
```

可以查看到输出结果...

```
_C:\worksp\yiibaiweb-core>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------
[INFO] Building yiibaiweb-core 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ yiibaiweb
[INFO] Deleting C:\worksp\yiibaiweb-core\target
[INFO] ------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
[INFO] Total time: 0.538 s
[INFO] Finished at: 2015-11-03T07:32:48+08:00
[INFO] Final Memory: 6M/77M
[INFO] ------------------------------------------------------------------
```

当“mvn clean”执行，在“target”文件夹中的一切都将被删除。

部署进行生产 要部署您的项目进行生产，它总是建议使用 “mvn clean package“, 以确保始终获得最新的部署。

# 使用Maven运行单元测试 - Maven教程

要通过Maven运行单元测试，发出此命令：

```
mvn test
```

这会在你的项目中运行整个单元测试。

## 案例学习

创建两个单元测试，并通过 Maven 的运行它。参见一个简单的 Java 测试类：

```
package com.yiibai.core;

public class App {
    public static void main(String[] args) {

        System.out.println(getHelloWorld());

    }

    public static String getHelloWorld() {

        return "Hello World";

    }

    public static String getHelloWorld2() {

        return "Hello World 2";

    }
}
```

## Unit Test 1

单元测试为getHelloWorld()方法。

```
package com.yiibai.core;

import junit.framework.Assert;
import org.junit.Test;

public class TestApp1 {

    @Test
    public void testPrintHelloWorld() {

        Assert.assertEquals(App.getHelloWorld(), "Hello World");

    }

}
```

## Unit Test 2

单元测试为getHelloWorld2()方法。

```
package com.yiibai.core;

import junit.framework.Assert;
import org.junit.Test;

public class TestApp2 {

    @Test
    public void testPrintHelloWorld2() {

        Assert.assertEquals(App.getHelloWorld2(), "Hello World 2")

    }

}
```

## 运行单元测试

使用Maven运行单元测试看见下面的例子。

示例 **1** 运行整个单元测试(TestApp1和TestApp2)，发出以下命令：

```
mvn test
```

示例 **2** 为了运行单个测试(TestApp1)，发出此命令：

```
mvn -Dtest=TestApp1 test [INFO] --- maven-compiler-plugin:3.1:compi
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testRe
ibai-core ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\worksp\yiibai-core\sr
ces
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompi
core ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ yiiba
[INFO] Surefire report directory: C:\worksp\yiibai-core\target\sure

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.yiibai.core.TestApp1
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.0

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] -------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] -------------------------------------------------------------
[INFO] Total time: 2.143 s
[INFO] Finished at: 2015-11-03T20:29:50+08:00
[INFO] Final Memory: 11M/114M
[INFO] -------------------------------------------------------------
```

示例 **3** 为了运行单个测试(TestApp2)，发出此命令：

```
mvn -Dtest=TestApp2 test
```

注意 欲了解更多“mvn test”的例子，请参考Maven测试插件文档。

# 将项目安装到Maven本地资源库 - Maven教程

在Maven中，可以使用“mvn install”打包项目，并自动部署到本地资源库，让其他开发人员使用它。

```
mvn install
```

注意， 当“install”在执行阶段，上述所有阶段 “validate“, “compile“, “test“, “package“, “integration-test“, “verify” 阶段, 包括目前的“install”阶段将被执行有序。请参阅Maven的生命周期细节。

## mvn install 示例

一个Java项目，具有以下 pom.xml 文件

*File : pom.xml*

```
_<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.yiibai.core</groupId>
  <artifactId>yiibai-core</artifactId>
  <packaging>jar</packaging>
  <version>99</version>
  <name>yiibai-core</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.4</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>_
```

基于以上pom.xml文件，在“mvn install”被执行， 它会打包项目为“yiibai-core-99.jar”文件，并复制到本地存储库。

警告 它总是建议运行“clean”和“install”在一起，让您能始终部署最新的项目到本地存储库。

```
mvn clean install
```

# 生成基于Maven的项目文档站点 - Maven教程

在Maven中，可以使用“mvn site”，为您的项目信息生成文档站点。

```
mvn site
```

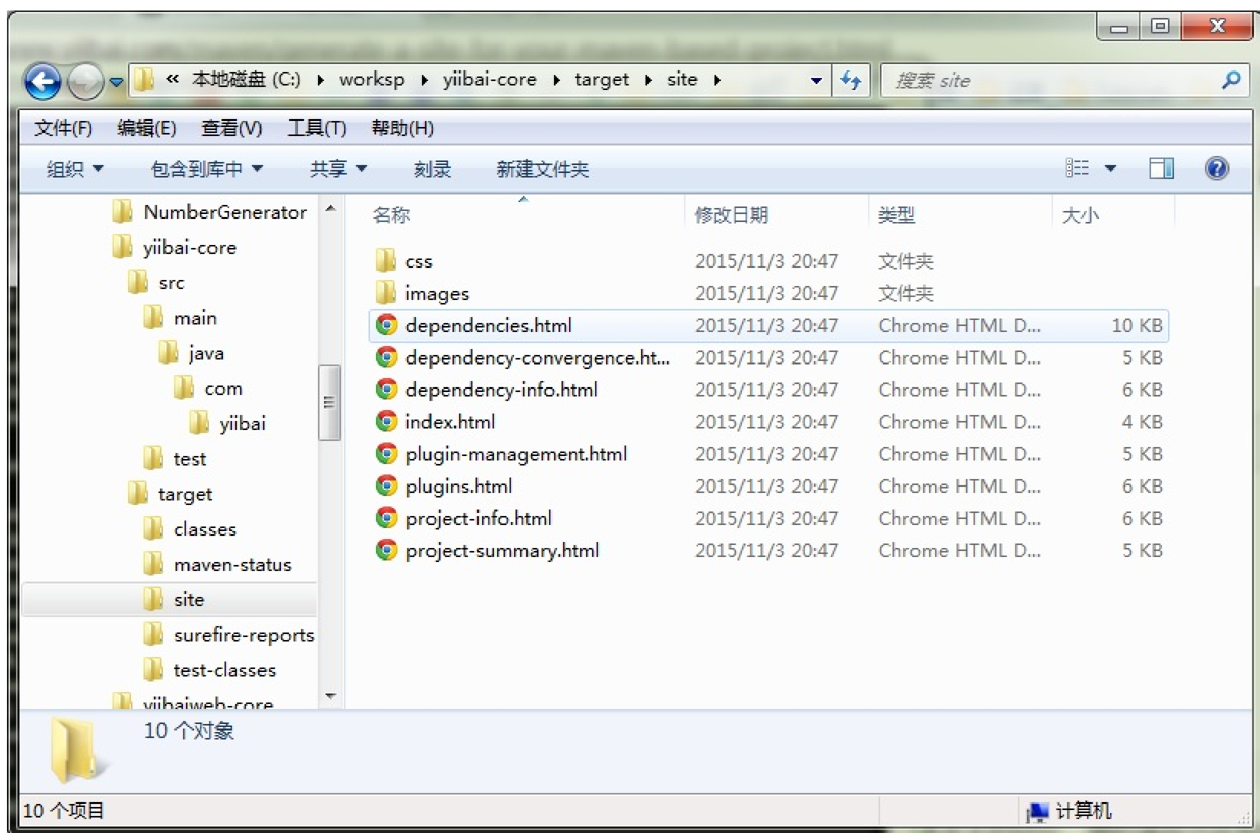
生成的网站是在项目的“target/site”文件夹中。

## mvn site 示例

请参见通过“mvn site”命令生成的文件列表。



文档页面的样本如下。




注意 就个人而言，我不喜欢这个功能了，因为它并没有带来多大的好处，想知道自己开发的项目的信息可以试试。

# 使用“mvn site-deploy”部署站点（WebDAV例子） - Maven教程

这里有一个指南，向您展示如何使用“mvn site:deploy”来自动部署生成的文档站点到服务器，这里通过WebDAV机制说明。

P.S 在这篇文章中，我们使用的是Apache服务器2.x的WebDAV功能。

## 1. 启用 WebDAV

请参见本指南，了解 如何启用WebDAV访问Apache 2.x服务器。

## 2. 配置在何处部署

在 pom.xml 中，配置在 “distributionManagement” 标签部署你的网站。

```
<distributionManagement>
    <site>
      <id>yiibaiserver</id>
      <url>dav:http://127.0.0.1/sites/</url>
    </site>
</distributionManagement>
```

注 “dav”前缀是HTTP协议之前添加的，这意味着通过WebDAV机制部署您的网站。或者，可以用“scp”取代它，如果您的服务器支持“scp”访问。

告诉Maven来使用“wagon-webdav-jackrabbit”扩展部署。

```
<build>
    <extensions>
        <extension>
            <groupId>org.apache.maven.wagon</groupId>
            <artifactId>wagon-webdav-jackrabbit</artifactId>
            <version>1.0-beta-7</version>
        </extension>
    </extensions>
</build>
```

**wagon-webdav** 一些人说可以使用“wagon-webdav”，但这不是我试了不能正常工作，所以这里用“wagon-webdav-jackrabbit”代替。

```
<extension>
            <groupId>org.apache.maven.wagon</groupId>
            <artifactId>wagon-webdav</artifactId>
            <version>1.0-beta-2</version>
</extension>
```

*pom.xml* 整个文件内容：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.yiibai.core</groupId>
  <artifactId>yiibai-core</artifactId>
  <packaging>jar</packaging>
  <version>1</version>
  <name>yiibai-core</name>
  <url>http://maven.apache.org</url>
  <build>
    <extensions>
        <extension>
            <groupId>org.apache.maven.wagon</groupId>
            <artifactId>wagon-webdav-jackrabbit</artifactId>
            <version>1.0-beta-7</version>
        </extension>
    </extensions>
  </build>
  <distributionManagement>
    <site>
      <id>yiibaiserver</id>
      <url>dav:http://127.0.0.1/sites/</url>
    </site>
  </distributionManagement>
</project>
```

## 3. 配置WebDAV身份验证

通常情况下，WebDAV是需要认证的访问。所以需要把相关的认证细节(用户名和密码)%MAVEN_PATH%/conf/settings.xml.

*File : settings.xml*

```
<servers>
    <server>
        <id>yiibaiserver</id>
        <username>admin</username>
        <password>123456</password>
    </server>
</servers>
```

**“yiibaiserver”** 是什么 **?** 在Maven的“的settings.xml”文件服务器ID将在“的pom.xml”文件被网站引用。

# 4. mvn site:deploy

“mvn site:deploy” 命令执行：

```
C:\worksp\yiibai-core>mvn site:deploy
... ...
Transfer finished. 11622 bytes copied in 0.021 seconds
十一月 03, 2015 9:00:07 下午 org.apache.commons.httpclient.auth.Auth
cessor selectAuthScheme
信息: digest authentication scheme selected
Uploading: .//project-info.html to http://127.0.0.1/sites/

##十一月 03, 2015 9:00:07 下午 org.apache.commons.httpclient.auth.Au
rocessor selectAuthScheme
信息: digest authentication scheme selected
##http://127.0.0.1/sites//./project-info.html - Status code: 201

Transfer finished. 11170 bytes copied in 0.035 seconds
十一月 03, 2015 9:00:07 下午 org.apache.commons.httpclient.auth.Auth
cessor selectAuthScheme
信息: digest authentication scheme selected
Uploading: .//project-summary.html to http://127.0.0.1/sites/

##十一月 03, 2015 9:00:07 下午 org.apache.commons.httpclient.auth.Au
rocessor selectAuthScheme
信息: digest authentication scheme selected
##http://127.0.0.1/sites//./project-summary.html - Status code: 20

Transfer finished. 10190 bytes copied in 0.021 seconds
http://127.0.0.1/sites/ - Session: Disconnecting
http://127.0.0.1/sites/ - Session: Disconnected
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 28.737 s
[INFO] Finished at: 2015-11-03T21:00:07+08:00
[INFO] Final Memory: 14M/156M
[INFO] ------------------------------------------------------------
```

所有站点文件夹和文件，在项目文件夹- “target/site”会被自动部署到服务器。

# 5. 输出

在本例中，可以通过这个网址访问该部署的站点：http://127.0.0.1/sites/， 见下图：




完成.

# 参考


1. http://maven.apache.org/plugins/maven-site-plugin/usage.html
2. http://mojo.codehaus.org/wagon-maven-plugin/usage.html
3. http://maven.apache.org/plugins/maven-site-plugin/deploy-mojo.html
4. http://maven.40175.n5.nabble.com/site-deploy-using-DAV-with-digest-auth-td125042.html
5. http://www.sonatype.com/books/maven-book/reference/site-generation-sect-deploy-site.html

# 部署基于Maven的war文件到Tomcat - Maven教程

在本教程中，我们将学习如何使用Maven的Tomcat插件打包并部署一个WAR文件到Tomcat(Tomcat的6和7。

要用到工具：

1. Maven 3
2. Tomcat 6.0.37
3. Tomcat 7.0.53

**Tomcat 7** 发布URL = *http://localhost:8080/manager/text* 命令 = mvn tomcat7:deploy

**Tomcat 6** 发布 URL = *http://localhost:8080/manager/* 命令 = mvn tomcat6:deploy

## 1. Tomcat 7 示例

这个例子说明了如何在Tomcat7打包并部署WAR文件。

1.1 Tomcat 认证 添加具有角色管理器GUI和管理脚本的用户。

%TOMCAT7_PATH%/conf/tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>

    <role rolename="manager-gui"/>
    <role rolename="manager-script"/>
    <user username="admin" password="password" roles="manager-gui,r

</tomcat-users>
```

**1.2 Maven** 认证 添加在上面Maven 设置文件的 Tomcat 用户，是之后Maven使用此用户来登录Tomcat服务器。

%MAVEN_PATH%/conf/settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<settings ...>
    <servers>

        <server>
            <id>TomcatServer</id>
            <username>admin</username>
            <password>password</password>
        </server>

    </servers>
</settings>
```

**1.3 Tomcat7 Maven** 插件 声明一个Maven的Tomcat插件。

pom.xml

```
    <plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
        <configuration>
            <url>http://localhost:8080/manager/text</url>
            <server>TomcatServer</server>
            <path>/yiibaiWebApp</path>
        </configuration>
    </plugin>
```

怎么运行的？ 在部署过程中，它告诉 Maven 通过部署 WAR 文件Tomcat服务器，"http://localhost:8080/manager/text" , 在路径"/yiibaiWebApp"上, 使用 "TomcatServer" (settings.xml) 用户名和密码来进行认证。

**1.4** 发布到**Tomcat** 以下的命令是用来操纵Tomcat WAR文件。

```
mvn tomcat7:deploy
mvn tomcat7:undeploy
mvn tomcat7:redeploy
```

示例

```
> mvn tomcat7:deploy

...
[INFO] Deploying war to http://localhost:8080/yiibaiWebApp
Uploading: http://localhost:8080/manager/text/deploy?path=%2Fyiiba
Uploaded: http://localhost:8080/manager/text/deploy?path=%2FyiibaiW

[INFO] tomcatManager status code:200, ReasonPhrase:OK
[INFO] OK - Deployed application at context path /yiibaiWebApp
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 8.507 s
[INFO] Finished at: 2015-08-05T11:35:25+08:00
[INFO] Final Memory: 28M/308M
[INFO] ------------------------------------------------------------
```

## 2. Tomcat 6 示例

这个例子说明了如何在Tomcat6打包和部署WAR文件。这些步骤和Tomcat7是一样的，只是部署URL和命令名称不同。

2.1 Tomcat 认证

%TOMCAT6_PATH%/conf/tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>

    <role rolename="manager-gui"/>
    <role rolename="manager-script"/>
    <user username="admin" password="password" roles="manager-gui,r

</tomcat-users>
```

**2.2 Maven 认证**

%MAVEN_PATH%/conf/settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<settings ...>
    <servers>

        <server>
            <id>TomcatServer</id>
            <username>admin</username>
            <password>password</password>
        </server>

    </servers>
</settings>
```

### 2.3 Tomcat6 Maven 插件

pom.xml

```
<plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat6-maven-plugin</artifactId>
        <version>2.2</version>
        <configuration>
            <url>http://localhost:8080/manager</url>
            <server>TomcatServer</server>
            <path>/yiibaiWebApp</path>
        </configuration>
    </plugin>
```

### 2.4 发布到Tomcat

```
mvn tomcat6:deploy
mvn tomcat6:undeploy
mvn tomcat6:redeploy
```

示例

```
> mvn tomcat6:deploy

...
[INFO] Deploying war to http://localhost:8080/yiibaiWebApp
Uploading: http://localhost:8080/manager/deploy?path=%2FyiibaiWebAp
Uploaded: http://localhost:8080/manager/deploy?path=%2FyiibaiWebApp

[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 22.652 s
[INFO] Finished at: 2014-08-05T12:18:54+08:00
[INFO] Final Memory: 30M/308M
[INFO] ------------------------------------------------------------
```

## 参考

1. Apache Tomcat 7 Manager App HOW-TO
2. Apache Tomcat 6 Manager App HOW-TO
3. Tomcat Maven Plugin
4. Tomcat Maven Plugin – Context Goals

# MyBatis教程

MyBatis 是支持普通 SQL 查询,存储过程和高级映射的优秀持久层框架。MyBatis 消除 了几乎所有的 JDBC 代码和参数的手工设置以及结果集的检索。MyBatis 使用简单的 XML 或注解用于配置和原始映射,将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java 对象)映射成数据库中的记录。

每个MyBatis应用程序主要都是使用SqlSessionFactory实例的，一个SqlSessionFactory实例可以通过SqlSessionFactoryBuilder获得。SqlSessionFactoryBuilder可以从一个xml配置文件或者一个预定义的配置类的实例获得。 用xml文件构建SqlSessionFactory实例是非常简单的事情。推荐在这个配置中使用类路径资源（classpath resource)，但你可以使用任何Reader实例，包括用文件路径或file://开头的url创建的实例。MyBatis有一个实用类----Resources，它有很多方法，可以方便地从类路径及其它位置加载资源。 MyBatis 最强大的特性之一就是它的动态语句功能。如果您以前有使用JDBC或者类似框架的经历，您就会明白把SQL语句条件连接在一起是多么的痛苦，要确保不能忘记空格或者不要在columns列后面省略一个逗号等。动态语句能够完全解决掉这些痛苦。尽管与动态SQL一起工作不是在开一个party，但是MyBatis确实能通过在任何映射SQL语句中

使用强大的动态SQL来改进这些状况。

```
动态SQL元素对于任何使用过JSTL或者类似于XML之类的文本处理器的人来说，都是非常熟
```

# MyBatis环境配置及入门 - MyBatis教程

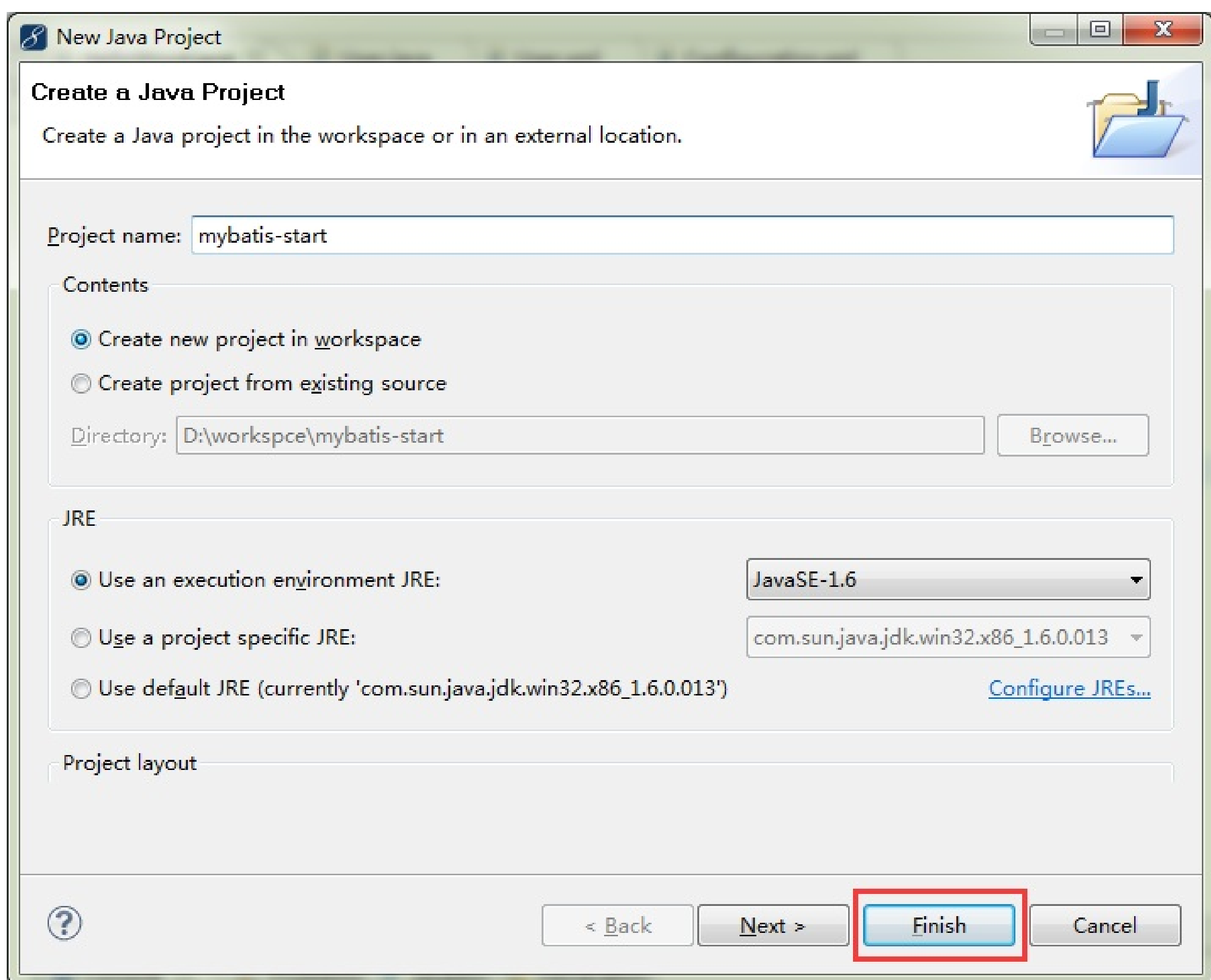
Mybatis 开发环境搭建，选择: MyEclipse8.5 版本，mysql 5.5, jdk 1.8, mybatis3.2.3.jar 包。这些软件工具均可以到各自的官方网站上下载。

整个过程在概如下，

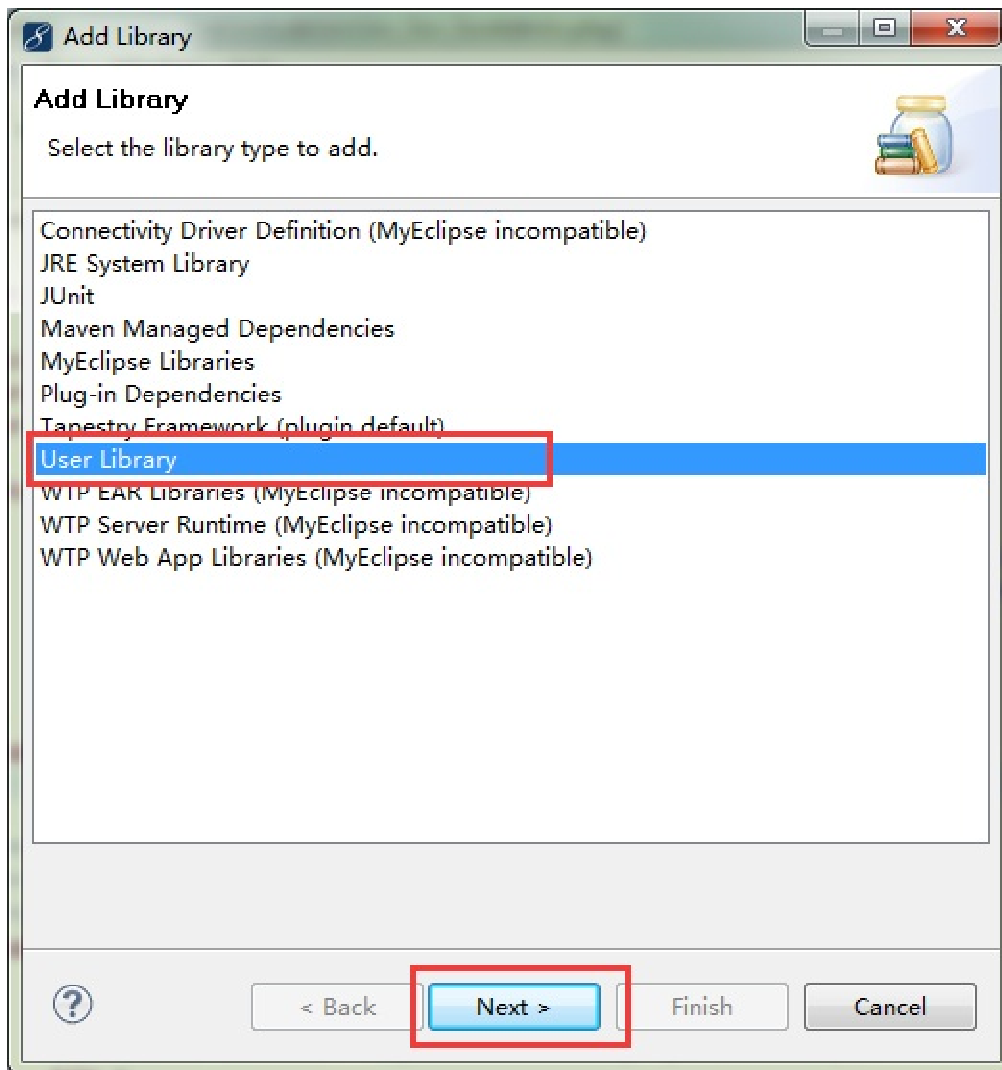
1. 在本教程中，建立 java 工程，但一般都是开发 web 项目，这个系列教程最后是 web，但这里为了方便学习，本教程前面建立的都是 java 工程。 2. 将 mybatis-3.2.3.jar，mysql-connector-java-5.1.25-bin.jar 创建两个用户自定库（User Libary）：mysql-connector 和 mybatis； 3. 创建 mysql 测试数据库和用户表, 注意，数据库使用的是 utf-8 编码。以解决不必要的中文乱码问题。

## 一、 创建 Java 工程

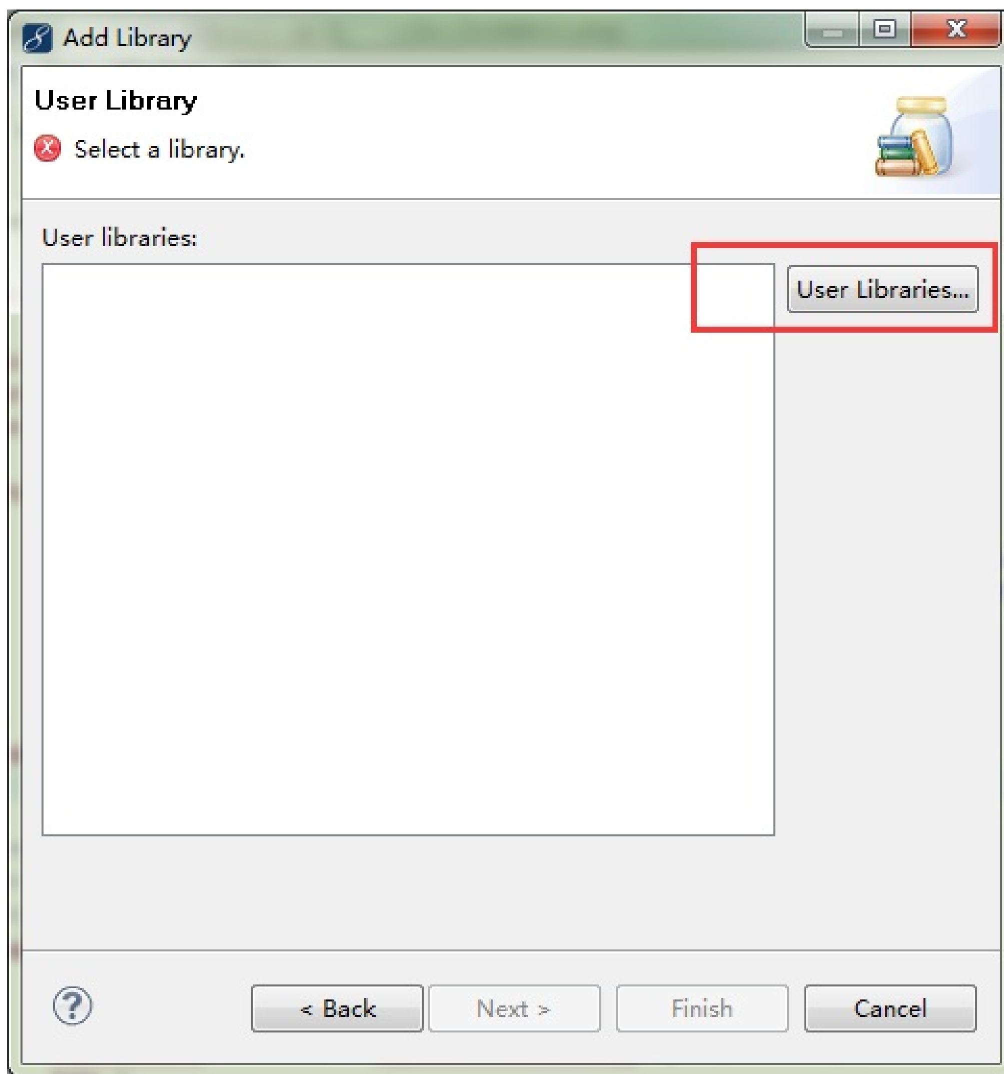
首先建立一个名字为 Helloword 的 java project。选择 "File" -> "New" ->"Java Project"，如下图所示：



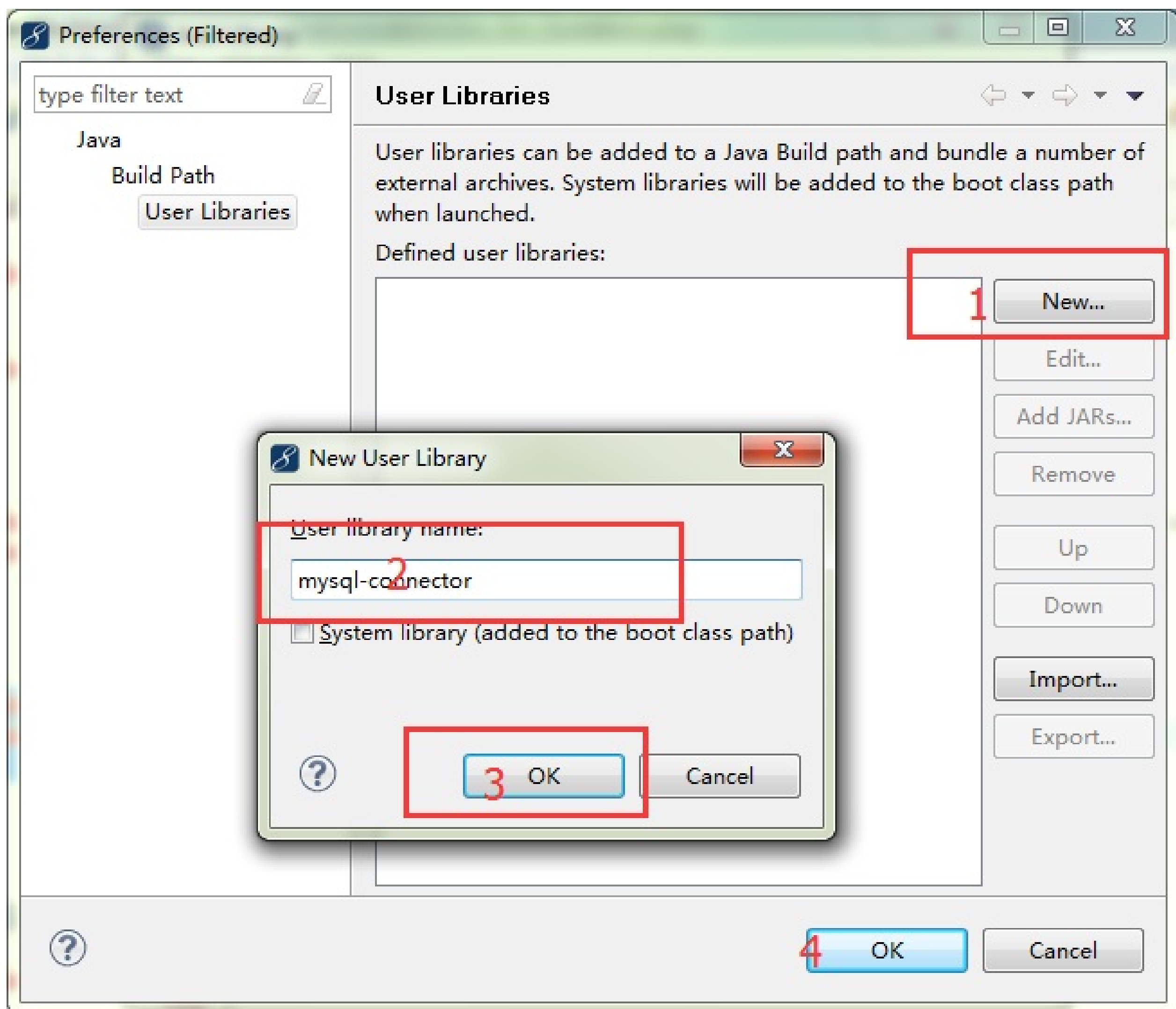
这样就创建了一个 Java 工程了，我们继续下一步。接下来我们在 mybatis-start 项目中加入两个所需的程序库：mysql-connector 和 mybatis，右键点击 "mybatis-start" 项目，从弹出的菜单中选择： "Build Path" -> "Add Libaries..."， 如下图所示：




从中选择“User Library"，点击“Next>"，创建两个类库为：mysql-connector 和 mybatis，如下所示：




点击“User Libararies...”，继续下一步，创建一个新的类库，如下图所示：

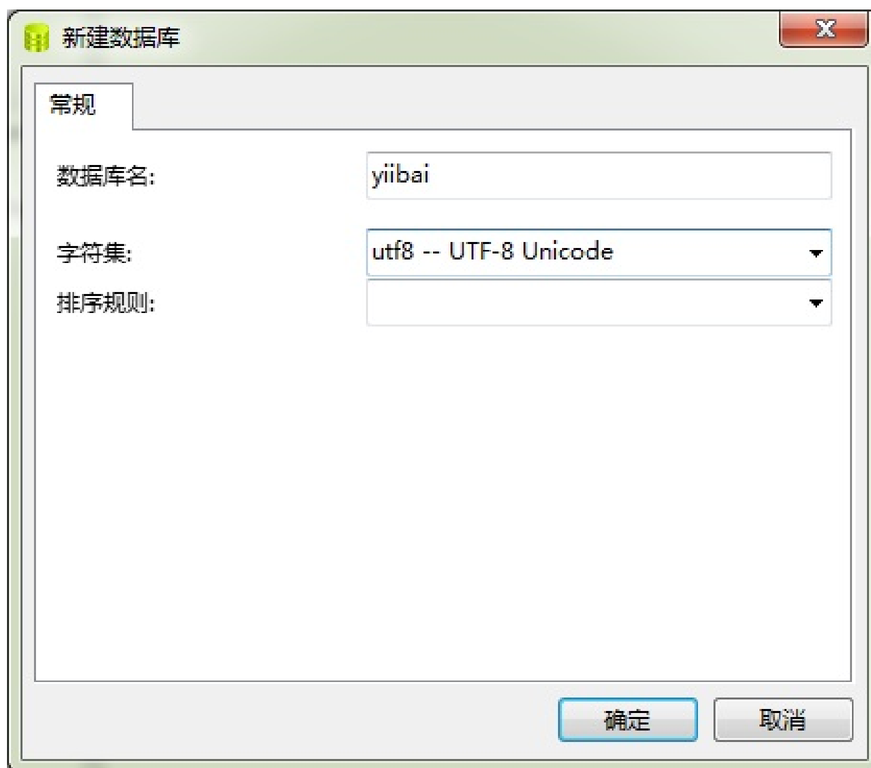



到此用户的一个类库创建完成，以相同的方式来创建另一个类库：

## 二、创建数据库和 User 表

创建所需的数据库和表，要创建的数据库是：yiibai，并在 yiibai 数据库创建一个表：user，如下图所示：

创建数据库：yiibai，使用 utf-8 编码。




接下我们创建一个表：user，并插入一条记录信息，其结构如下所示：

```
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(64) NOT NULL DEFAULT '',
  `dept` varchar(254) NOT NULL DEFAULT '',
  `website` varchar(254) DEFAULT '',
  `phone` varchar(16) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'yiibai', 'Tech', 'http://www.yiiba
```

## 三、创建 Mybatis 配置文件

到此为止，前期准备工作就完成了。下面开始真正配置 mybatis-start 项目。设置 mybatis 配置文件: Configure.xml, 在 src/config 目录下建立此文件，内容如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.mybatis.models.Use
    </typeAliases>
    <environments default="development">
        <environment id="development">
        <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                            <property name="driver" value="com.mysc
                <property name="url" value="jdbc:mysql://127.0.0.1
                <property name="username" value="root" />
                <property name="password" value="" />
                       </dataSource>
        </environment>
    </environments>

    <mappers>
         <!-- // power by http://www.yiibai.com -->
         <mapper resource="com/yiibai/mybatis/models/User.xml" />
    </mappers>
</configuration>
```

## 四、创建实体类和映射文件

首先创建一个包：com.yiibai.mybatis.models，并在下创建与数据库表对应的 User.java 类及其映射文件：User.xml，详细如下图所示：

```
package com.yiibai.mybatis.models;

public class User {
    private int id;
    private String name;
    private String dept;
    private String phone;
    private String website;

    public String getWebsite() {
        return website;
    }
    public void setWebsite(String website) {
        this.website = website;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDept() {
        return dept;
    }
    public void setDept(String dept) {
        this.dept = dept;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }

}
```

同时建立这个 User 类对应的映射文件 User.xml，详细如下代码所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.yiibai.mybatis.models.UserMapper">
    <select id="GetUserByID" parameterType="int" resultType="User">
        select * from `user` where id = #{id}
    </select>
</mapper>
```

下面是对这几个配置文件一点解释说明： 1、配置文件 Configure.xml 是 mybatis 用来建立 sessionFactory，里面主要包含了数据库连接相关内容，还有 java 类所对应的别名，比如： 这个别名非常重要，在具体的类的映射中，比如：User.xml 中 resultType 就是对应这个。要保持一致，这里的 resultType 还有另外单独的定义方式，后面学习到我们再详细介绍说明。 2、Configure.xml 里面 的是包含要映射的类的 xml 配置文件。 3、在User.xml 文件里面主要是定义各种 SQL 语句，以及这些语句的参数，以及要返回的类型等等。

## 五、运行程序测试结果

在 src 源码目录下建立 一个类叫作：HelloWord, 来运行测试配置环境是否成功，具体代码如下示：

```
import java.io.Reader;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.yiibai.mybatis.models.*;

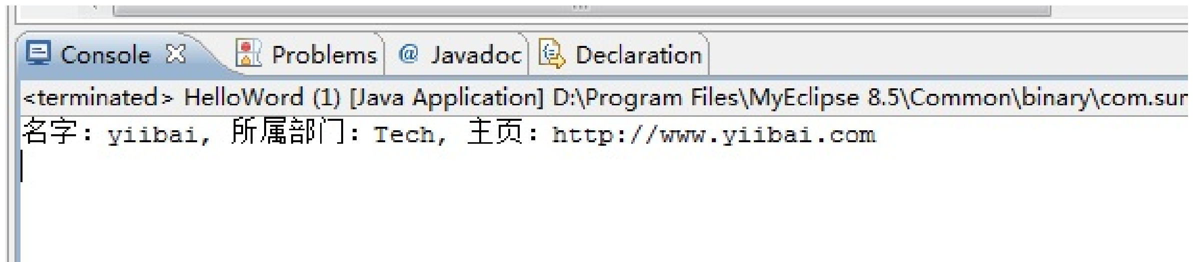
/**
 *
 * @author yiibai
 * @copyright http://www.yiibai.com
 * @date 2015/09/22
 */
public class HelloWord {
    private static SqlSessionFactory sqlSessionFactory;
    private static Reader reader;

    static {
        try {
            reader = Resources.getResourceAsReader("config/Configu
            sqlSessionFactory = new SqlSessionFactoryBuilder().bui
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static SqlSessionFactory getSession() {
        return sqlSessionFactory;
    }
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SqlSession session = sqlSessionFactory.openSession();
        try {
            User user = (User) session.selectOne(
                    "com.yiibai.mybatis.models.UserMapper.GetUserBy
            if(user!=null){
                String userInfo = "名字："+user.getName()+", 所属部门
                System.out.println(userInfo);
            }
        } finally {
            session.close();
        }
    }

}
```

现在运行这个程序，不是得到查询结果了？正确的输出结果应该如下：

```
名字：yiibai, 所属部门：Tech, 主页：http://www.yiibai.com
```




恭喜你，环境搭建配置成功，在接下来章节，我们将学习 Mybatis 的操作方式：增删改查。

Jar 包下载：http://pan.baidu.com/s/1bnyRJ9H



# Mybatis接口注解 - MyBatis教程

在上一章中，我们已经搭建了 myeclipse,mybatis,mysql 的开发环境，并且实现了 mybatis 的一个简单的查询。要注意的是，这种方式是用 SqlSession 实例来直接执行已映射的 SQL 语句：

session.selectOne("com.yiibai.mybatis.models.UserMapper.getUserByID", 1)，但是还有比这更简单的方法，而且是更好的方法，使用合理描述参数和SQL语句返回值的接口(比如：IUser.class)，这样现在就可以至此那个更简单，更安全的代码，没有容易发生的字符串文字和转换的错误，下面是详细过程:

## 1、创建一个接口：**IUser**，并在其中声明对应的操作方法

在 src 源码目录下创建一个包：com.yiibai.mybatis.dao，并建立接口类 IUser 及一个方法, 在方法上面，我们使用了一个 SQL注释，内容如下：

```
package com.yiibai.mybatis.dao;

import org.apache.ibatis.annotations.Select;

import com.yiibai.mybatis.models.User;
/**
 * @author yiibai.com
 */
public interface IUser {
     @Select("select * from user where id= #{id}")
     public User getUserByID(int id);
}
```

请注意，这里面代码有一个方法名 getUserByID 必须与 User.xml 里面配置的 select 的 id 对应(<select id="getUserByID")同名。

## 2、创建对应映射接口 **SQL** 语句

首先配置 MyBatis 所需的数据连接文件，这里创建一个文件：src/config/Configure.xml，其内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.mybatis.models.Use
    </typeAliases>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Drive
                <property name="url" value="jdbc:mysql://127.0.0.1
                <property name="username" value="root" />
                <property name="password" value="" />
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <!-- // power by http://www.yiibai.com -->
        <mapper resource="com/yiibai/mybatis/models/User.xml" />
    </mappers>
</configuration>
```

在包：com.yiibai.mybatis.models 下创建一个 User.java 类文件，与上一节中 User 类代码相同，这里只是拷贝过来，User.java 具体的内容如下：

```
package com.yiibai.mybatis.models;

public class User {
    private int id;
    private String name;
    private String dept;
    private String phone;
    private String website;

    public String getWebsite() {
        return website;
    }
    public void setWebsite(String website) {
        this.website = website;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDept() {
        return dept;
    }
    public void setDept(String dept) {
        this.dept = dept;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }

}
```

与 User.java 对应的 XML 配置文件：User.xml 内容如下所示（注：在这一小节小，此功能用不上）：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.yiibai.mybatis.models.UserMapper">
    <select id="getUserByID" parameterType="int" resultType="User">
        select * from `user` where id = #{id}
    </select>
</mapper>
```

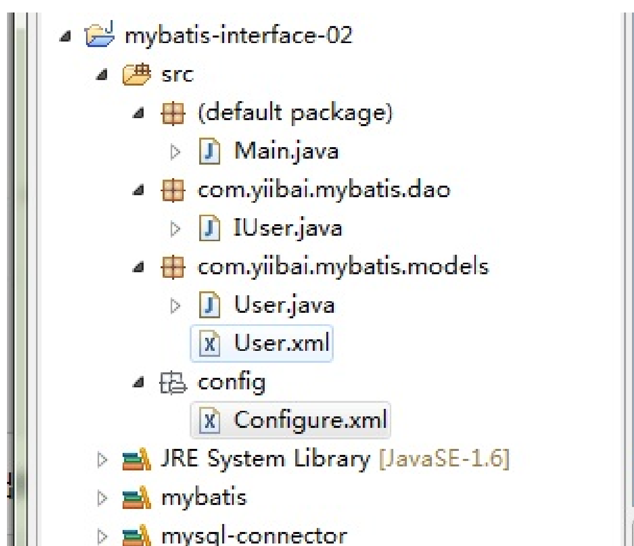
这里要注意的是，IUser.java 有一个方法名 getUserByID 必须与 User.xml 里面配置的 select 的 id 对应(<select id="getUserByID")同名。

## 3、测试接口映射

我们在 src 这个目录下创建一个类：Main.java，用来测试整个配置和程序运行结果，有关 Main.java 的代码详细内容如下：

```
名字：New name, 所属部门：Tech, 主页：http://www.yiibai.com
```

最后补充，整个工程 mybatis-interface-02 的结构如下图所示：




注意：最后还有一个重要说明，需要导入两个 jar 库：mysql-connector 和 mybatis3.jar。在上一小节中，有介绍怎么创建用户库。

# Mybatis增删改查（CURD） - MyBatis教程

前面的小节我们已经讲到用接口的方式编程。使用这种方式，需要注意的一个地方就是，在User.xml 配置文件中，mapper namespace="com.yiibai.mybatis.inter.IUser" ，命名空间对应非常重要，名称不能有错，必须与我们定义的 package 和 接口一致。如果不一致就会出错，这一章主要在上一讲基于接口编程的基础上完成如下操作: 1. 使用 mybatis 查询用户数据(读取用户列表) 2. 使用 mybatis 增加用户数据 3. 使用 mybatis 更新用户数据 4. 使用 mybatis 删除用户数据

查询数据，前面已经讲过简单的查询单个用户数据，在这里将查询出用户列表，

要查询出列表，也就是返回 List, 在我们这个例子中也就是 List<User> , 要以这种方式返回数据，需要在 User.xml 里面配置返回的类型 resultMap, 注意不是 resultType, 而这个resultMap 所对应的应该是我们自己配置。

## 1、创建工程并配置所需环境

我们首先来创建一个工程：mybatis-curd-03，与第一节中介绍的环境配置一样，加入所需的 jar 包：mysql-connector 和 mybatis3.jar。配置 src/config/Configure.xml，其文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.mybatis.models.Use
    </typeAliases>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Drive
                <property name="url" value="jdbc:mysql://127.0.0.1
                <property name="username" value="root" />
                <property name="password" value="" />
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <!-- // power by http://www.yiibai.com -->
        <mapper resource="com/yiibai/mybatis/models/User.xml" />
    </mappers>
</configuration>
```

## 2、创建 Java 类和接口

在这里需要创建一个类和一个接口：User.java类和IUser.java接口，User.java类位于包 com.yiibai.mybatis.models 下，User.java类代码内容如下：

```
package com.yiibai.mybatis.models;

public class User {
    private int id;
    private String name;
    private String dept;
    private String phone;
    private String website;

    public String getWebsite() {
        return website;
    }
    public void setWebsite(String website) {
        this.website = website;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDept() {
        return dept;
    }
    public void setDept(String dept) {
        this.dept = dept;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }

}
```

IUser.java接口位于包 com.yiibai.mybatis.dao 下，IUser.java接口代码内容如下：

```
package com.yiibai.mybatis.dao;

import java.util.List;

import org.apache.ibatis.annotations.Select;

import com.yiibai.mybatis.models.User;
/**
 *
 * @author yiibai
 *
 */
public interface IUser {
    //@Select("select * from user where id= #{id}")
    //public User getUserByID(int id);
    public List<User> getUserList();

    public void insertUser(User user);

    public void updateUser(User user);

    public void deleteUser(int userId);

    public User getUser(int id);
}
```

这里还需要一个XML文件，与前一小节中一样，使用的是 User.xml，在这其中，我们分别对应了增删改查的操作（每一个操作的 ID 对应于IUser接口的方法），其内容如下：

```
import java.io.Reader;
import java.text.MessageFormat;
import java.util.List;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.yiibai.mybatis.dao.IUser;
import com.yiibai.mybatis.models.User;

public class Main {
    private static SqlSessionFactory sqlSessionFactory;
    private static Reader reader;

    static {
        try {
            reader = Resources.getResourceAsReader("config/Configu
            sqlSessionFactory = new SqlSessionFactoryBuilder().buil
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static SqlSessionFactory getSession() {
        return sqlSessionFactory;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SqlSession session = sqlSessionFactory.openSession();
        try {
            //User user = (User) session.selectOne(
            //        "com.yiibai.mybatis.models.UserMapper.getUser
            IUser iuser = session.getMapper(IUser.class);
            // 用户数据列表
            //getUserList();
            // 插入数据
            //testInsert();
            //testUpdate();

            // 删除数据
            testDelete();

        } finally {
            session.close();
        }
    }
    //
    public static void testInsert()
    {
        try
        {
            // 获取Session连接
            SqlSession session = sqlSessionFactory.openSession();
            // 获取Mapper
            IUser userMapper = session.getMapper(IUser.class);
            System.out.println("Test insert start...");
            // 执行插入
            User user = new User();
            user.setId(0);
            user.setName("Google");
            user.setDept("Tech");
            user.setWebsite("http://www.google.com");
            user.setPhone("120");
            userMapper.insertUser(user);
            // 提交事务
            session.commit();
```

```
            // 显示插入之后User信息
            System.out.println("
After insert");
            getUserList();
            System.out.println("Test insert finished...");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

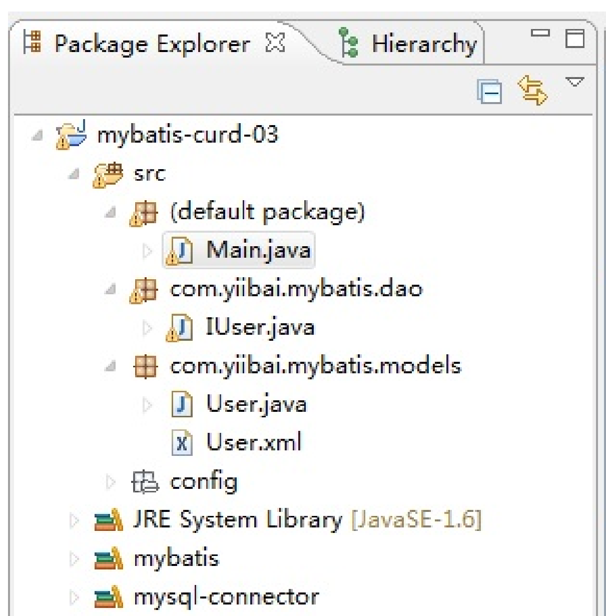
    // 获取用户列表
    public static void getUserList(){
        try
        {
            SqlSession session = sqlSessionFactory.openSession();
            IUser iuser = session.getMapper(IUser.class);
            // 显示User信息
            System.out.println("Test Get start...");
            printUsers(iuser.getUserList());
            System.out.println("Test Get finished...");
        }catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    public static void testUpdate()
    {
        try
        {
            SqlSession session = sqlSessionFactory.openSession();
            IUser iuser = session.getMapper(IUser.class);
            System.out.println("Test update start...");
            printUsers(iuser.getUserList());
            // 执行更新
            User user = iuser.getUser(1);
            user.setName("New name");
            iuser.updateUser(user);
            // 提交事务
            session.commit();
            // 显示更新之后User信息
            System.out.println("
After update");
            printUsers(iuser.getUserList());
            System.out.println("Test update finished...");
        }catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    // 删除用户信息
    public static void testDelete()
```

```
    {
        try
        {
            SqlSession session = sqlSessionFactory.openSession();
            IUser iuser = session.getMapper(IUser.class);
            System.out.println("Test delete start...");
            // 显示删除之前User信息
            System.out.println("Before delete");
            printUsers(iuser.getUserList());
            // 执行删除
            iuser.deleteUser(3);
            // 提交事务
            session.commit();
            // 显示删除之后User信息
            System.out.println("
After delete");
            printUsers(iuser.getUserList());
            System.out.println("Test delete finished...");
        }catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    /**
     *
     * 打印用户信息到控制台
     *
     * @param users
     */
    private static void printUsers(final List<User> users)
    {
        int count = 0;

        for (User user : users)
        {
            System.out.println(MessageFormat.format("============
            System.out.println("User Id: " + user.getId());
            System.out.println("User Name: " + user.getName());
            System.out.println("User Dept: " + user.getDept());
            System.out.println("User Website: " + user.getWebsite(
        }
    }
}
```

执行以上程序，如果没有问题，应该能正确输出。那么这里所有增删改查都完成了，需要注意的是在增加，更改，删除的时候需要调用 session.commit() 来提交事务，这样才会真正对数据库进行操作提交保存，否则操作没有提交到数据中。 到此为止，简单的单表操作已经完成了，接下来在下一节中将会讲解多表联合查询，以及结果集的选取。 如遇到不明白的问题，请留言评论。

最后，附上工程结果图，如下：


Package Explorer
Hierarchy
mybatis-curd-03
src
(default package)
Main.java
com.yiibai.mybatis.dao
IUser.java
com.yiibai.mybatis.models
User.java
User.xml
config
JRE System Library [JavaSE-1.6]
mybatis
mysql-connector

# Mybatis表关联一对多 - MyBatis教程

有了前面几章的基础，对一些简单的应用是可以处理的，但在实际项目中，经常是关联表的查询，比如：最常见到的多对一，一对多等。这些查询是如何处理的呢，这一讲就讲这个问题。前面几篇教程中介绍的都是单表映射的一些操作，然而在我们的实际项目中往往是用到多表映射。在Java实体对象对中，一对多可以根据List和Set来实现，两者在mybitis中都是通过collection标签来配合来加以实现。这篇介绍的是多表中的多对一表关联查询。

应用场景：首先根据用户 ID 读取一个用户信息，然后再读取这个用户所发布贴子(post)。

## 1、先做一些准备工作

我们首先在创建一个 java 工程，工程名称为：mybatis04-one2many(下载)，还需要创建两张表，它们分别是用户表 user，和帖子表 post，一个户用户可以有多个帖子。 **user**表的结构和数据：

```
-- ----------------------------
-- Table structure for `user`
-- ----------------------------
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(64) NOT NULL DEFAULT '',
  `mobile` int(10) unsigned NOT NULL DEFAULT '0',
  `created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'yiibai', '100', '2015-09-23 20:11
```

帖子表 **post** 的结构和数据：

```
-- ----------------------------
-- Table structure for `post`
-- ----------------------------
CREATE TABLE `post` (
  `post_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `userid` int(10) unsigned NOT NULL,
  `title` varchar(254) NOT NULL DEFAULT '',
  `content` text,
  `created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`post_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of post
-- ----------------------------
INSERT INTO `post` VALUES ('1', '1', 'MyBatis关联数据查询', '在实际项目
INSERT INTO `post` VALUES ('2', '1', 'MyBatis开发环境搭建', '为了方便学
INSERT INTO `post` VALUES ('3', '2', '这个是别人发的', 'content,内容..
```

从上面应该看出，这几个帖子对应的 userid 都是1，所以需要用户表 user 里面有 id=1 的数据。可以修改成满足自己条件的数据，按照orm的规则，表肯定需要一个对象与之对应，所以我们增加一个 Post 类。

## 2、创建表对应的 JavaBean 对象

这个例子中，我们需要在包 com.yiibai.pojo 下创建两个类，它们分别是：User.java 和 Post.java，我们一个一个地来看它们的代码，User.java 类的代码如下：

```
package com.yiibai.pojo;

import java.io.Serializable;
import java.util.Date;
import java.util.List;

public class User implements Serializable{
    private int id;
    private String username;
    private String mobile;
    private List<Post> posts;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getMobile() {
        return mobile;
    }

    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

    public List<Post> getPosts() {
        return posts;
    }
    public void setPosts(List<Post> posts) {
        this.posts = posts;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + username + "]";
    }

}
```

Post.java 类的代码如下：

```
package com.yiibai.pojo;

import java.io.Serializable;

public class Post implements Serializable{
    private int id;
    private User user;
    private String title;
    private String content;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
}
```

## 3、配置文件

在这一章节中，要用到的配置文件有两个，一个是 mybatis 的主配置文件：src/config/Configure.xml 和 User.java对应的配置文件 User.xml，我们先来看看 src/config/Configure.xml，其详细配置信息如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.pojo.User" />
        <typeAlias alias="Post" type="com.yiibai.pojo.Post" />
    </typeAliases>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Drive
                <property name="url" value="jdbc:mysql://127.0.0.1
                <property name="username" value="root" />
                <property name="password" value="" />
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <!-- // power by http://www.yiibai.com -->
        <mapper resource="com/yiibai/pojo/User.xml" />
    </mappers>
</configuration>
```

这里需要注意的是 这个标签内容，它就是用于定义一个 JavaBean 类的别名，如将 com.yiibai.pojo.User 简写为 User，可以认为 com.yiibai.pojo.User 就是 User，User 就是 com.yiibai.pojo.User 。

另外一个配置文件 User.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.yiibai.userMaper">
    <!-- User 级联文章查询 方法配置 (一个用户对多个文章)  -->

    <resultMap type="User" id="resultUserMap">
        <result property="id" column="user_id" />
        <result property="username" column="username" />
        <result property="mobile" column="mobile" />
        <collection property="posts" ofType="com.yiibai.pojo.Post"
            <id property="id" column="post_id" javaType="int" jdbcT
            <result property="title" column="title" javaType="strin
            <result property="content" column="content" javaType="s
        </collection>
    </resultMap>

    <select id="getUser" resultMap="resultUserMap" parameterType="i
        SELECT u.*,p.*
        FROM user u, post p
        WHERE u.id=p.userid AND id=#{user_id}
  </select>

</mapper>
```

## 4、测试程序运行

到这里，整个工作准备得已经差不多了，我们创建一个主类来测试上面程序，在 src 下创建一个 Main.java，代码如下：

```
import java.io.Reader;
import java.text.MessageFormat;
import java.util.List;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.yiibai.pojo.Post;
import com.yiibai.pojo.User;

public class Main {
    private static SqlSessionFactory sqlSessionFactory;
    private static Reader reader;

    static {
        try {
            reader = Resources.getResourceAsReader("config/Configu
            sqlSessionFactory = new SqlSessionFactoryBuilder().buil
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static SqlSessionFactory getSession() {
        return sqlSessionFactory;
    }

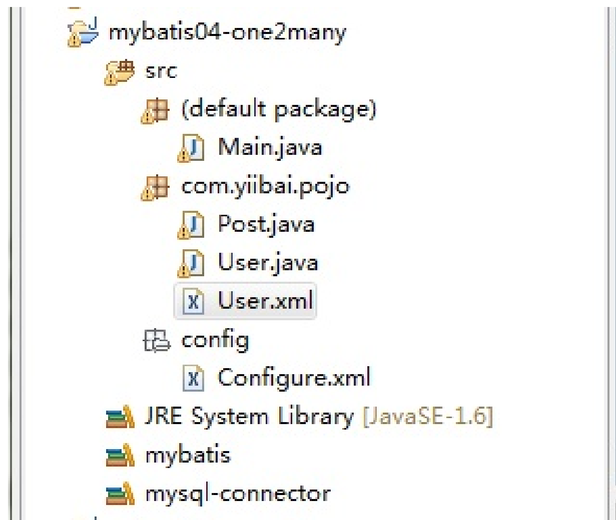
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SqlSession session = sqlSessionFactory.openSession();
        try {
            int userid = 1;
            User user = session.selectOne("com.yiibai.userMaper.ge
             System.out.println("username: "+user.getUsername()+",
                     List<Post> posts = user.getPosts();
                     for(Post p : posts) {
                         System.out.println("Title:" + p.getTitle(
                         System.out.println("Content:" + p.getConte
                      }
         } finally {
            session.close();
         }
    }

}
```

输出结果如下：

```
username: yiibai,
Title:MyBatis关联数据查询
Content:在实际项目中，经常使用关联表的查询，比如：多对一，一对多等。这些查询是
Title:MyBatis开发环境搭建
Content:为了方便学习，这里直接建立java 工程，但一般都是开发 Web 项目。
```

附工程目录结构图如下：



# Mybatis表关联多对一 - MyBatis教程

在上章的 一对多 中，我们已经学习如何在 Mybatis 中关联多表，但在实际项目中也是经常使用 多对一 的情况，这些查询是如何处理的呢，在这一节中我们来学习它。多表映射的多对一关系要用到 mybitis 的 association 来加以实现。这篇介绍的是多表中的多对一表关联查询。

应用场景：首先根据帖子 ID 读取一个帖子信息，然后再读取这个帖子所属的用户信息。

## 1、先做一些准备工作

我们首先在创建一个 java 工程，工程名称为：mybatis05-many2one(下载)，还需要创建两张表，它们分别是用户表 user，和帖子表 post，一个户用户可以有多个帖子。

user表的结构和数据：

```
-- ----------------------------
-- Table structure for `user`
-- ----------------------------
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(64) NOT NULL DEFAULT '',
  `mobile` int(10) unsigned NOT NULL DEFAULT '0',
  `created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'yiibai', '100', '2015-09-23 20:11
```

帖子表 post 的结构和数据：

```
-- ----------------------------
-- Table structure for `post`
-- ----------------------------
CREATE TABLE `post` (
  `post_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `userid` int(10) unsigned NOT NULL,
  `title` varchar(254) NOT NULL DEFAULT '',
  `content` text,
  `created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`post_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of post
-- ----------------------------
INSERT INTO `post` VALUES ('1', '1', 'MyBatis关联数据查询', '在实际项目
INSERT INTO `post` VALUES ('2', '1', 'MyBatis开发环境搭建', '为了方便学
INSERT INTO `post` VALUES ('3', '2', '这个是别人发的', 'content,内容..
```

从上面应该看出，这几个帖子对应的 userid 都是1，所以需要用户表 user 里面有 id=1 的数据。可以修改成满足自己条件的数据，按照 orm 的规则，表肯定需要一个对象与之对应，所以我们增加一个 Post 类。

## 2、创建表对应的 JavaBean 对象

这个例子中，我们需要在包 com.yiibai.pojo 下创建两个类，它们分别是：User.java 和 Post.java，我们一个一个地来看它们的代码，User.java 类的代码如下：

```
package com.yiibai.pojo;

import java.io.Serializable;
import java.util.Date;
import java.util.List;

public class User implements Serializable{
    private int id;
    private String username;
    private String mobile;
    private List<Post> posts;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getMobile() {
        return mobile;
    }

    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

    public List<Post> getPosts() {
        return posts;
    }
    public void setPosts(List<Post> posts) {
        this.posts = posts;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + username + "]";
    }

}
```

Post.java 类的代码如下：

```
package com.yiibai.pojo;

import java.io.Serializable;

public class Post implements Serializable{
    private int id;
    private User user;
    private String title;
    private String content;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
}
```

## 3、配置文件

在这一章节中，要用到的配置文件有两个，一个是 mybatis 的主配置文件：src/config/Configure.xml 和 User.java对应的配置文件 User.xml，我们先来看看 src/config/Configure.xml，其详细配置信息如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.pojo.User" />
        <typeAlias alias="Post" type="com.yiibai.pojo.Post" />
    </typeAliases>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Drive
                <property name="url" value="jdbc:mysql://127.0.0.1
                <property name="username" value="root" />
                <property name="password" value="" />
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <!-- // power by http://www.yiibai.com -->
        <mapper resource="com/yiibai/pojo/User.xml" />
    </mappers>
</configuration>
```

这里需要注意的是 这个标签内容，它就是用于定义一个 JavaBean 类的别名，如将 com.yiibai.pojo.User 简写为 User，可以认为 com.yiibai.pojo.User 就是 User，User 就是 com.yiibai.pojo.User 。

另外一个配置文件 User.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.yiibai.userMaper">
    <!-- User 级联文章查询 方法配置 (多个文章对一个用户)  -->

    <resultMap type="Post" id="resultPostsMap">
        <result property="id" column="post_id" />
        <result property="title" column="title" />
        <result property="content" column="content" />
        <association property="user" javaType="User">
            <id property="id" column="userid"/>
            <result property="username" column="username"/>
            <result property="mobile" column="mobile"/>
        </association>
    </resultMap>

    <select id="getPosts" resultMap="resultPostsMap" parameterType=
        SELECT u.*,p.*
        FROM user u, post p
        WHERE u.id=p.userid AND p.post_id=#{post_id}
  </select>

</mapper>
```

注：在上面的配置文件中，使用到了一个 <association>标签，关联对应的 User 类。

## 4、测试程序运行

到这里，整个工作准备得已经差不多了，我们创建一个主类来测试上面程序，在 src 下创建一个 Main.java，代码如下：

```
import java.io.Reader;
import java.text.MessageFormat;
import java.util.List;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.yiibai.pojo.Post;
import com.yiibai.pojo.User;

public class Main {
    private static SqlSessionFactory sqlSessionFactory;
    private static Reader reader;

    static {
        try {
            reader = Resources.getResourceAsReader("config/Configu
            sqlSessionFactory = new SqlSessionFactoryBuilder().bui
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static SqlSessionFactory getSession() {
        return sqlSessionFactory;
    }

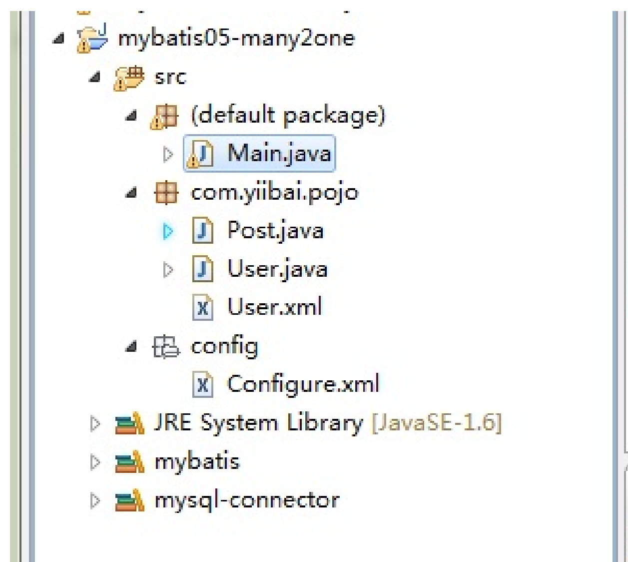
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SqlSession session = sqlSessionFactory.openSession();
        try {
            int postId = 1;
            Post post = session.selectOne("com.yiibai.userMaper.ge
            System.out.println("title: "+post.getTitle());
            System.out.println("userName: "+post.getUser().getUser
        } finally {
            session.close();
        }
    }

}
```

输出结果如下：

```
title: MyBatis关联数据查询
userName: yiibai
```

附工程目录结构图如下：



# Mybatis 多对多 - MyBatis教程

在前面的章节中，我们学习了一对多，多对一的关系，现在我们来看看 Mybatis 中的多对多应用。

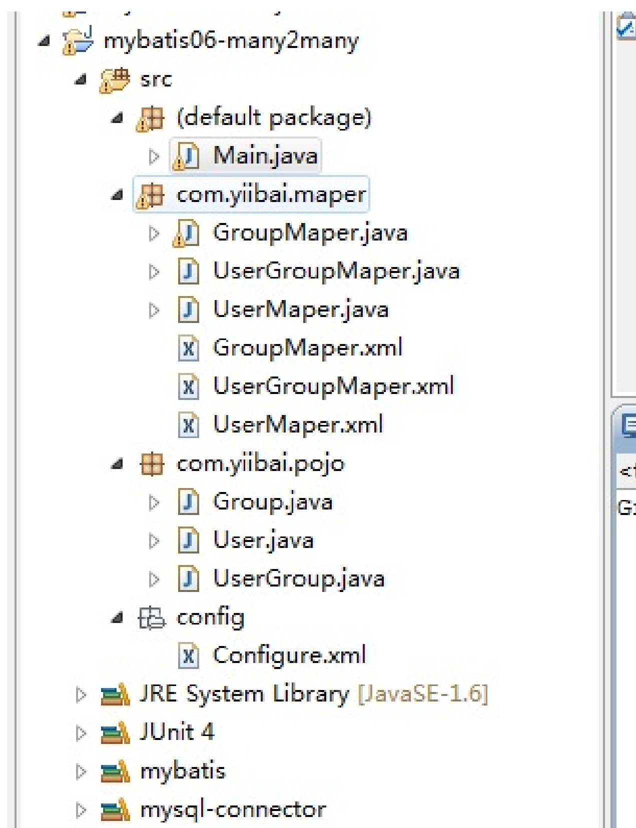
mybatis3.0 添加了association和collection标签专门用于对多个相关实体类数据进行级联查询，但仍不支持多个相关实体类数据的级联保存和级联删除操作。因此在进行实体类多对多映射表设计时，需要专门建立一个关联对象类对相关实体类的关联关系进行描述。下文将以“User”和“Group"两个实体类之间的多对多关联映射为例进行CRUD操作。

## 1、 应用场景

假设项目中存在用户和用户组，从一个用户读取出它所在的用户组，从一个用户组也知道这个组内的所有用户信息。

## 2、先做一些准备工作

我们首先在创建一个 java 工程，工程名称为：mybatis06-many2many(下载)，还需要创建三张表，它们分别是用户表 user，用户组表 group 和 用户组映射表 user_group，一个户用户可以在多个用户组中，一个用户组中有多个用户。项目工程结构如下：



user表的结构和数据：

```
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(64) NOT NULL DEFAULT '',
  `mobile` varchar(16) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');
INSERT INTO `user` VALUES ('2', 'User-name-1', '13838009988');
```

用户组 group 表的结构和数据：

```
CREATE TABLE `group` (
  `group_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `group_name` varchar(254) NOT NULL DEFAULT '',
  PRIMARY KEY (`group_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of group
-- ----------------------------
INSERT INTO `group` VALUES ('1', 'Group-1');
INSERT INTO `group` VALUES ('2', 'Group-2');
```

用户组映射表 user_group 的结构和数据：

```
CREATE TABLE `user_group` (
  `user_id` int(10) unsigned NOT NULL DEFAULT '0',
  `group_id` int(10) unsigned NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user_group
-- ----------------------------
INSERT INTO `user_group` VALUES ('1', '1');
INSERT INTO `user_group` VALUES ('2', '1');
INSERT INTO `user_group` VALUES ('1', '2');
```

从上面应该看出，用户ID为1同时在用户组ID为 1 和 2 中，而用户ID为 2 仅在一个用户组ID为1中。

## 2、创建表对应的 JavaBean 对象

这个例子中，我们需要在包 com.yiibai.pojo 下创建三个类，它们分别是：User.java 、Group.java 和 UserGroup.java，让我们一个一个地来看它们的代码，User.java 类的代码如下：

```
package com.yiibai.pojo;

import java.util.List;

/**
 * @describe: User
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class User {
    private int id;
    private String username;
    private String mobile;
    private List<Group> groups;
    public List<Group> getGroups() {
        return groups;
    }
    public void setGroups(List<Group> groups) {
        this.groups = groups;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getMobile() {
        return mobile;
    }
    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

}
```

Group.java 类的代码如下：

```
package com.yiibai.pojo;

import java.util.List;

/**
 * @describe: Group
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class Group {
    private int groupId;
    private String groupName;
    private List<User> users;

    public List<User> getUsers() {
        return users;
    }
    public void setUsers(List<User> users) {
        this.users = users;
    }
    public int getGroupId() {
        return groupId;
    }
    public void setGroupId(int groupId) {
        this.groupId = groupId;
    }
    public String getGroupName() {
        return groupName;
    }
    public void setGroupName(String groupName) {
        this.groupName = groupName;
    }

}
```

UserGroup.java 类(用户和用户组的关系映射)的代码如下：

```
package com.yiibai.pojo;

public class UserGroup {
    private int userId;
    private int groupId;
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public int getGroupId() {
        return groupId;
    }
    public void setGroupId(int groupId) {
        this.groupId = groupId;
    }

}
```

## 3、配置文件

在这一章节中，要用到的配置文件有四个，一个是 mybatis 的主配置文件：src/config/Configure.xml， 另外就是上面三个Bean类对应的配置文件，如，User.java 对应的配置文件 User.xml，等，我们先来看看 src/config/Configure.xml，其详细配置信息如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.pojo.User" />
        <typeAlias alias="UserGroup" type="com.yiibai.pojo.UserGrou
        <typeAlias alias="Group" type="com.yiibai.pojo.Group" />
    </typeAliases>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Drive
                <property name="url" value="jdbc:mysql://127.0.0.1
                <property name="username" value="root" />
                <property name="password" value="" />
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <!-- // power by http://www.yiibai.com -->
        <mapper resource="com/yiibai/maper/UserMaper.xml" />
        <mapper resource="com/yiibai/maper/GroupMaper.xml" />
        <mapper resource="com/yiibai/maper/UserGroupMaper.xml" />
    </mappers>
</configuration>
```

Group.java 对应的配置文件 src/com/yiibai/maper/Group.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.yiibai.maper.GroupMaper">

    <parameterMap type="Group" id="parameterGroupMap">
        <parameter property="groupId"/>
        <parameter property="groupName"/>
    </parameterMap>
    <insert id="insertGroup" parameterMap="parameterGroupMap">
        INSERT INTO `group` (group_name)
        VALUES(#{groupName});
    </insert>

    <resultMap type="Group" id="resultGroupMap_1">
        <result property="id" column="id" />
        <result property="groupName" column="group_name" />
        <collection property="users" column="group_id"
            select="com.yiibai.maper.UserGroupMaper.getUsersByGrou
    </resultMap>
    <select id="getGroup" resultMap="resultGroupMap_1"
        parameterType="int">
        SELECT *
        FROM `group`
        WHERE group_id=#{id}
    </select>
</mapper>
```

User.java 对应的配置文件 src/com/yiibai/maper/User.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.yiibai.maper.UserMaper">
    <parameterMap type="User" id="parameterUserMap">
        <parameter property="id"/>
        <parameter property="username"/>
        <parameter property="mobile"/>
    </parameterMap>

    <insert id="insertUser" parameterMap="parameterUserMap">
        INSERT INTO user(username,mobile)
        VALUES(#{username},#{mobile});
    </insert>

    <resultMap type="User" id="resultUser">
        <result property="id" column="group_id"/>
        <result property="name" column="name"/>
        <collection property="groups" column="id" select="com.yiiba
    </resultMap>

    <select id="getUser" resultMap="resultUser" parameterType="int'
        SELECT *
        FROM user
        WHERE id=#{id}
    </select>
</mapper>
```

UserGroup.java 对应的配置文件 src/com/yiibai/maper/UserGroup.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.yiibai.maper.UserGroupMaper">
    <parameterMap type="UserGroup" id="parameterUserGroupMap">
        <parameter property="userId"/>
        <parameter property="groupId"/>
    </parameterMap>

    <insert id="insertUserGroup"  parameterMap="parameterUserGroupM
        INSERT INTO user_group(user_id, group_id)
        VALUES(#{userId},#{groupId})
    </insert>

    <!-- 根据一个用户组ID,查看这个用户组下的所有用户 -->
    <resultMap type="User" id="resultUserMap_2">
        <result property="id" column="id"/>
        <result property="username" column="username"/>
        <result property="mobile" column="mobile"/>
    </resultMap>

    <select id="getUsersByGroupId" resultMap="resultUserMap_2" para
        SELECT u.*, ug.group_id
        FROM user u, user_group ug
        WHERE u.id=ug.user_id AND ug.group_id=#{group_id}
    </select>

    <!-- 根据一个用户ID,查看这个用户所对应的组-->
    <resultMap type="Group" id="resultGroupMap_2">
        <result property="groupId" column="group_id"/>
        <result property="groupName" column="group_name"/>
    </resultMap>

    <select id="getGroupsByUserId" resultMap="resultGroupMap_2" par
        SELECT g.*, u.user_id
        FROM group g, user_group u
        WHERE g.group_id=u.group_id AND u.user_id=#{user_id}
    </select>
</mapper>
```

注：在上面的配置文件中，使用到了 <association>和 <clollection>标签，关联对应的 User 类和 Group类。

## 4、测试程序运行

到这里，整个工作准备得已经差不多了，我们创建一个主类来测试上面程序，在 src 下创建一个 Main.java，代码如下：

```
import java.io.Reader;
```

```
import java.text.MessageFormat;
import java.util.List;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.yiibai.maper.GroupMaper;
import com.yiibai.maper.UserGroupMaper;
import com.yiibai.maper.UserMaper;
import com.yiibai.pojo.Group;
import com.yiibai.pojo.User;
import com.yiibai.pojo.UserGroup;

public class Main {
    private static SqlSessionFactory sqlSessionFactory;
    private static Reader reader;

    static {
        try {
            reader = Resources.getResourceAsReader("config/Configur
            sqlSessionFactory = new SqlSessionFactoryBuilder().buil
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static SqlSessionFactory getSession() {
        return sqlSessionFactory;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // testAddGroup();
        // testAddUser();
        // testAddUserGroup();
        testGetGroupAndUsers();

    }

    public static void testGetGroupAndUsers() {
        UserGroup userGroup = new UserGroup();
        SqlSession session = sqlSessionFactory.openSession();
        try {
            GroupMaper groupMaper = session.getMapper(GroupMaper.cl
            Group group = groupMaper.getGroup(1);
            System.out.println("Group => " + group.getGroupName());
            List<User> users = group.getUsers();
            for (User user : users) {
```

```
                System.out.println("\t:" + user.getId() + "\t"
                        + user.getUsername());
            }
        } finally {
            session.close();
        }
    }

    public static void testAddUserGroup() {
        UserGroup userGroup = new UserGroup();
        userGroup.setGroupId(1);
        userGroup.setUserId(2);
        SqlSession session = sqlSessionFactory.openSession();
        try {
            UserGroupMaper userGroupMaper = session
                    .getMapper(UserGroupMaper.class);
            userGroupMaper.insertUserGroup(userGroup);

            session.commit();
        } finally {
            session.close();
        }

    }

    public static void testAddUser() {
        // TODO Auto-generated method stub
        SqlSession session = sqlSessionFactory.openSession();
        try {
            User user = new User();
            user.setUsername("User-name-1");
            user.setMobile("13838009988");
            UserMaper userMaper = session.getMapper(UserMaper.class
            userMaper.insertUser(user);
            session.commit();
            // System.out.println(user.getGroupId());
        } finally {
            session.close();
        }
    }

    public static void testAddGroup() {
        // TODO Auto-generated method stub
        SqlSession session = sqlSessionFactory.openSession();
        try {
            Group group = new Group();
            group.setGroupName("用户组-1");
            GroupMaper groupMapper = session.getMapper(GroupMaper.c
            groupMapper.insertGroup(group);
            session.commit();
            System.out.println(group.getGroupId());
        } finally {
            session.close();
```

```
        }
    }
}
```

运行上述程序，得出结果：

```
Group => Group-1
    :1    yiibai
    :2    User-name-1
```

# Mybatis与Spring集成 - MyBatis教程

在前面的教程文章中，前面讲到有关 mybatis 连接数据库，然后进行进行数据增删改查，以及多表联合查询的例子，但很多的项目中，通常会用 spring 这个粘合剂来管理 datasource 等。充分利用 spring 基于接口的编程，以及aop ,ioc 带来的方便。用 spring 来管理 mybatis 与管理 hibernate 有很多类似的地方。在这一节中，我们重点介绍数据源管理以及 bean 的配置。

整个Mybatis与Spring集成示例要完成的步骤如下：

1、示例功能描述

2、创建工程

3、数据库表结构及数据记录

4、实例对象

5、配置文件

6、测试执行，输出结果

## 1、示例功能描述

在本示例中，需要完成这样的一个简单功能，即，指定一个用户（ID=1），查询出这个用户的基本信息，并关联查询这个用户的所有订单。

## 2、创建工程

首先创建一个工程的名称为：mybatis07-spring，在 src 源代码目录下建立文件夹 config，并将原来的 mybatis 配置文件 Configuration.xml 移动到这个文件夹中, 并在 config 文家夹中建立 Spring 配置文件：applicationContext.xml。工程结构目录如下：



## 3、数据库表结构及数据记录

在本示例中，用到两个表：用户表和订单表，其结构和数据记录如下：

```
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(64) NOT NULL DEFAULT '',
  `mobile` varchar(16) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');
INSERT INTO `user` VALUES ('2', 'saya', '13838009988');
```

订单表结构和数据如下：

```
CREATE TABLE `order` (
  `order_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_id` int(10) unsigned NOT NULL DEFAULT '0',
  `order_no` varchar(16) NOT NULL DEFAULT '',
  `money` float(10,2) unsigned DEFAULT '0.00',
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of order
-- ----------------------------
INSERT INTO `order` VALUES ('1', '1', '1509289090', '99.90');
INSERT INTO `order` VALUES ('2', '1', '1519289091', '290.80');
INSERT INTO `order` VALUES ('3', '1', '1509294321', '919.90');
INSERT INTO `order` VALUES ('4', '1', '1601232190', '329.90');
INSERT INTO `order` VALUES ('5', '1', '1503457384', '321.00');
INSERT INTO `order` VALUES ('6', '1', '1598572382', '342.00');
INSERT INTO `order` VALUES ('7', '1', '1500845727', '458.00');
INSERT INTO `order` VALUES ('8', '1', '1508458923', '1200.00');
INSERT INTO `order` VALUES ('9', '1', '1504538293', '2109.00');
INSERT INTO `order` VALUES ('10', '1', '1932428723', '5888.00');
INSERT INTO `order` VALUES ('11', '1', '2390423712', '3219.00');
INSERT INTO `order` VALUES ('12', '1', '4587923992', '123.00');
INSERT INTO `order` VALUES ('13', '1', '4095378812', '421.00');
INSERT INTO `order` VALUES ('14', '1', '9423890127', '678.00');
INSERT INTO `order` VALUES ('15', '1', '7859213249', '7689.00');
INSERT INTO `order` VALUES ('16', '1', '4598450230', '909.20');
```

## 4、实例对象

用户表和订单表分别对应两个实例对象，分别是：User.java 和 Order.java，它们都在 com.yiibai.pojo 包中。

User.java代码内容如下：

```
package com.yiibai.pojo;

import java.util.List;

/**
 * @describe: User
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class User {
    private int id;
    private String username;
    private String mobile;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getMobile() {
        return mobile;
    }
    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

}
```

Order.java代码内容如下：

```
package com.yiibai.pojo;

/**
 * @describe: Order - 订单
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class Order {
    private int orderId;
    private String orderNo;
    private float money;
    private int userId;
    private User user;

    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public int getOrderId() {
        return orderId;
    }
    public void setOrderId(int orderId) {
        this.orderId = orderId;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public String getOrderNo() {
        return orderNo;
    }
    public void setOrderNo(String orderNo) {
        this.orderNo = orderNo;
    }
    public float getMoney() {
        return money;
    }
    public void setMoney(float money) {
        this.money = money;
    }

}
```

## 5、配置文件

这个实例中有三个重要的配置文件，它们分别是：applicationContext.xml，Configuration.xml 以及 UserMaper.xml。

applicationContext.xml 配置文件里最主要的配置：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.pojo.User" />
        <typeAlias alias="Order" type="com.yiibai.pojo.Order" />
    </typeAliases>
    <!-- Mybatis和Spring 集成之后,这些可以完全删除（注释掉）,数据库连接的管
    <!--
        <environments default="development"> <environment id="devel
        <transactionManager type="JDBC"/> <dataSource type="POOLED'
        name="driver" value="com.mysql.jdbc.Driver"/> <property nar
        value="jdbc:mysql://127.0.0.1:3306/yiibai?characterEncoding
        <property name="username" value="root"/> <property name="pa
        value=""/> </dataSource> </environment> </environments>
    -->
    <mappers>
        <mapper resource="com/yiibai/maper/UserMaper.xml" />
    </mappers>
</configuration>
```

配置文件 Configuration.xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:conte
    xsi:schemaLocation="
            http://www.springframework.org/schema/beans http://www.
            http://www.springframework.org/schema/aop http://www.sp
            http://www.springframework.org/schema/context http://ww
            http://www.springframework.org/schema/jee http://www.sp
            http://www.springframework.org/schema/tx http://www.spr
    default-autowire="byName" default-lazy-init="false">

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataS
        <property name="driverClassName" value="com.mysql.jdbc.Driv
        <property name="url"
            value="jdbc:mysql://127.0.0.1:3306/yiibai?characterEnco
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessi
        <!--dataSource属性指定要用到的连接池-->
        <property name="dataSource" ref="dataSource" />
        <!--configLocation属性指定mybatis的核心配置文件-->
        <property name="configLocation" value="config/Configuration
    </bean>

    <bean id="userMaper" class="org.mybatis.spring.mapper.MapperFac
        <!--sqlSessionFactory属性指定要用到的SqlSessionFactory实例-->
        <property name="sqlSessionFactory" ref="sqlSessionFactory"
        <!--mapperInterface属性指定映射器接口，用于实现此接口并生成映射器对
        <property name="mapperInterface" value="com.yiibai.maper.Us
    </bean>
</beans>
```

UserMaper.xml 用于定义查询和数据对象映射，其内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.yiibai.maper.UserMaper">

        <!-- 为了返回list 类型而定义的returnMap -->
    <resultMap type="User" id="resultUser">
        <id column="id" property="id" />
        <result column="username" property="username" />
        <result column="mobile" property="mobile" />
    </resultMap>

    <!-- User 联合 Order 查询 方法的配置 (多对一的方式)  -->
    <resultMap id="resultUserOrders" type="Order">
        <id property="orderId" column="order_id" />
        <result property="orderNo" column="order_no" />
        <result property="money" column="money" />
        <result property="userId" column="user_id" />

        <association property="user" javaType="User">
            <id property="id" column="id" />
            <result property="username" column="username" />
            <result property="mobile" column="mobile" />
        </association>
    </resultMap>

    <select id="getUserOrders" parameterType="int" resultMap="resul
       SELECT u.*,o.* FROM `user` u, `order` o
              WHERE u.id=o.user_id AND u.id=#{id}
    </select>

    <select id="getUserById" resultMap="resultUser" parameterType="
        SELECT *
        FROM user
        WHERE id=#{id}
    </select>
</mapper>
```

## 6、测试执行，输出结果

我们创建一个测试类为：Main.java，就在 src 目录中。其代码如下：

```
import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.maper.UserMaper;
import com.yiibai.pojo.Order;
import com.yiibai.pojo.User;

/**
 * Description
 * @author yiibai
 * @date 2015-4-12
 * @copyright http://www.yiibai.com
 * @email yiibai.com@gmai.com
 * @version 1.0
 */

public class Main {

    private static ApplicationContext ctx;

    static {
        ctx = new ClassPathXmlApplicationContext(
                "config/applicationContext.xml");
    }

    public static void main(String[] args) {
        UserMaper userMaper = (UserMaper) ctx.getBean("userMaper");
        // 测试id=1的用户查询，可根据数据库中的情况修改.
        User user = userMaper.getUserById(1);
        System.out.println("获取用户 ID=1 的用户名："+user.getUsername

        // 得到文章列表测试
        System.out.println("得到用户id为1的所有订单列表:");
        System.out.println("=====================================
        List<Order> orders = userMaper.getUserOrders(1);

        for (Order order : orders) {
            System.out.println("订单号："+order.getOrderNo() + ", 订
        }

    }

}
```

运行结果如下：

```
log4j:WARN No appenders could be found for logger (org.springframew
log4j:WARN Please initialize the log4j system properly.
获取用户 ID=1 的用户名：yiibai
得到用户id为1的所有订单列表:
=============================================
订单号：1509289090, 订单金额：99.9
订单号：1519289091, 订单金额：290.8
订单号：1509294321, 订单金额：919.9
订单号：1601232190, 订单金额：329.9
订单号：1503457384, 订单金额：321.0
订单号：1598572382, 订单金额：342.0
订单号：1500845727, 订单金额：458.0
订单号：1508458923, 订单金额：1200.0
订单号：1504538293, 订单金额：2109.0
订单号：1932428723, 订单金额：5888.0
订单号：2390423712, 订单金额：3219.0
订单号：4587923992, 订单金额：123.0
订单号：4095378812, 订单金额：421.0
订单号：9423890127, 订单金额：678.0
订单号：7859213249, 订单金额：7689.0
订单号：4598450230, 订单金额：909.2
```

代码下载：Mybatis与Spring集成

# MyBatis整合Spring MVC - MyBatis教程

前面几篇文章已经讲到了mybatis与spring 的集成。目前主流的Web MVC框架，除了Struts这个主力外，还有Spring MVC，主要是由于 Spring MVC 配置比较简单，使用起来也十分明了，非常灵活，与Spring 集成较好，对 RESTful API 的支持也比struts 要好。所以Spring MVC在一定程度上有一定的优势。 MyBatis是ibatis的升级版，作为hibernate的老对手，它是一个可以自定义SQL、存储过程和高级映射的持久层框架。 与Hibernate 的主要区别就是 Mybatis 是半自动化的，而 Hibernate 是全自动的，所以当应用需求越来越复杂的时候，自动化的 SQL 显得比较笨拙。 经常搭框架的人应该都清楚，框架搭建的核心就是配置文件。

在这里我们需要创建 web 工程。今天将直接用 mybatis与Spring mvc 的方式集成起来，源码在本文结尾处下载.主要有以下几个方面的配置。

整个**Mybatis**与**Spring MVC** 示例要完成的步骤如下：

1、示例功能描述

2、创建工程

3、数据库表结构及数据记录

4、实例对象

5、配置文件

6、测试执行，输出结果

## 1、示例功能描述

在本示例中，需要使用 MyBatis和Spring MVC整合完成这样的一个简单功能，即指定一个用户（ID=1），查询出这个用户关联的所有订单。

## 2、创建工程

首先创建一个工程的名称为：mybatis07-spring-mvc，在 src 源代码目录下建立文件夹 config，并将原来的 mybatis 配置文件 Configuration.xml 移动到这个文件夹中, 并在 config 文家夹中建立 Spring 配置文件：applicationContext.xml。工程结构目录如下：



## 3、数据库表结构及数据记录

在本示例中，用到两个表：用户表和订单表，其结构和数据记录如下：

```
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(64) NOT NULL DEFAULT '',
  `mobile` varchar(16) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');
INSERT INTO `user` VALUES ('2', 'saya', '13838009988');
```

订单表结构和数据如下：

```
CREATE TABLE `order` (
  `order_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_id` int(10) unsigned NOT NULL DEFAULT '0',
  `order_no` varchar(16) NOT NULL DEFAULT '',
  `money` float(10,2) unsigned DEFAULT '0.00',
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of order
-- ----------------------------
INSERT INTO `order` VALUES ('1', '1', '1509289090', '99.90');
INSERT INTO `order` VALUES ('2', '1', '1519289091', '290.80');
INSERT INTO `order` VALUES ('3', '1', '1509294321', '919.90');
INSERT INTO `order` VALUES ('4', '1', '1601232190', '329.90');
INSERT INTO `order` VALUES ('5', '1', '1503457384', '321.00');
INSERT INTO `order` VALUES ('6', '1', '1598572382', '342.00');
INSERT INTO `order` VALUES ('7', '1', '1500845727', '458.00');
INSERT INTO `order` VALUES ('8', '1', '1508458923', '1200.00');
INSERT INTO `order` VALUES ('9', '1', '1504538293', '2109.00');
INSERT INTO `order` VALUES ('10', '1', '1932428723', '5888.00');
INSERT INTO `order` VALUES ('11', '1', '2390423712', '3219.00');
INSERT INTO `order` VALUES ('12', '1', '4587923992', '123.00');
INSERT INTO `order` VALUES ('13', '1', '4095378812', '421.00');
INSERT INTO `order` VALUES ('14', '1', '9423890127', '678.00');
INSERT INTO `order` VALUES ('15', '1', '7859213249', '7689.00');
INSERT INTO `order` VALUES ('16', '1', '4598450230', '909.20');
```

## 4、实例对象

用户表和订单表分别对应两个实例对象，分别是：User.java 和 Order.java，它们都在 com.yiibai.pojo 包中。

User.java代码内容如下：

```
package com.yiibai.pojo;

import java.util.List;

/**
 * @describe: User
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class User {
    private int id;
    private String username;
    private String mobile;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getMobile() {
        return mobile;
    }
    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

}
```

Order.java代码内容如下：

```
package com.yiibai.pojo;

/**
 * @describe: User - 订单
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class Order {
    private int orderId;
    private String orderNo;
    private float money;
    private int userId;
    private User user;

    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public int getOrderId() {
        return orderId;
    }
    public void setOrderId(int orderId) {
        this.orderId = orderId;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public String getOrderNo() {
        return orderNo;
    }
    public void setOrderNo(String orderNo) {
        this.orderNo = orderNo;
    }
    public float getMoney() {
        return money;
    }
    public void setMoney(float money) {
        this.money = money;
    }

}
```

## 5、配置文件

这个实例中有三个重要的配置文件，它们分别是：applicationContext.xml，Configuration.xml 以及 UserMaper.xml。

applicationContext.xml 配置文件里最主要的配置：

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:conte
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="
            http://www.springframework.org/schema/beans http://www.
            http://www.springframework.org/schema/aop http://www.sp
            http://www.springframework.org/schema/context http://ww
            http://www.springframework.org/schema/jee http://www.sp
            http://www.springframework.org/schema/tx http://www.spr
    default-autowire="byName" default-lazy-init="false">

    <!--本示例采用DBCP连接池，应预先把DBCP的jar包复制到工程的lib目录下。 --
    <context:property-placeholder location="classpath:/config/datab

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataS
        destroy-method="close" p:driverClassName="com.mysql.jdbc.Dr
        p:url="jdbc:mysql://127.0.0.1:3306/yiibai?characterEncoding
        p:username="root" p:password="" p:maxActive="10" p:maxIdle=
    </bean>

    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransa
        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessi
        <!--dataSource属性指定要用到的连接池-->
        <property name="dataSource" ref="dataSource" />
        <!--configLocation属性指定mybatis的核心配置文件-->
        <property name="configLocation" value="classpath:config/Con
        <!-- 所有配置的mapper文件 -->
        <property name="mapperLocations" value="classpath*:com/yiib
    </bean>

    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"
        <property name="basePackage" value="com.yiibai.maper" />
    </bean>
</beans>
```

配置文件 Configuration.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.pojo.User"/>
        <typeAlias alias="Order" type="com.yiibai.pojo.Order"/>
    </typeAliases>
    <!-- 与spring 集成之后,这些可以完全删除,数据库连接的管理交给 spring 去'
    <!--
    <environments default="development">
        <environment id="development">
        <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
            <property name="driver" value="com.mysql.jdbc.Driver"/>
            <property name="url" value="jdbc:mysql://127.0.0.1:3306
            <property name="username" value="root"/>
            <property name="password" value="password"/>
            </dataSource>
        </environment>
    </environments>
    -->

    <!-- 这里交给sqlSessionFactory 的 mapperLocations属性去得到所有配置
    <!--
    <mappers>
        <mapper resource="com/yihaomen/mapper/User.xml"/>
    </mappers>
    -->

</configuration>
```

UserMaper.xml 用于定义查询和数据对象映射，其内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.yiibai.maper.UserMaper">

        <!-- 为了返回list 类型而定义的returnMap -->
    <resultMap type="User" id="resultUser">
        <id column="id" property="id" />
        <result column="username" property="username" />
        <result column="mobile" property="mobile" />
    </resultMap>

    <!-- User 联合 Order 查询 方法的配置 (多对一的方式)  -->
    <resultMap id="resultUserOrders" type="Order">
        <id property="orderId" column="order_id" />
        <result property="orderNo" column="order_no" />
        <result property="money" column="money" />
        <result property="userId" column="user_id" />

        <association property="user" javaType="User">
            <id property="id" column="id" />
            <result property="username" column="username" />
            <result property="mobile" column="mobile" />
        </association>
    </resultMap>

    <select id="getUserOrders" parameterType="int" resultMap="resul
       SELECT u.*,o.* FROM `user` u, `order` o
              WHERE u.id=o.user_id AND u.id=#{id}
    </select>

    <select id="getUserById" resultMap="resultUser" parameterType='
        SELECT *
        FROM user
        WHERE id=#{id}
    </select>
</mapper>
```

## 6、测试执行，输出结果

我们创建一个控制器类在包 com.yiibai.controller 下，类的名称为：UserController.java，其代码如下：

```
package com.yiibai.controller;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.yiibai.maper.UserMaper;
import com.yiibai.pojo.Order;

/**
 * @describe: 读取一个用户下的所有订单
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */

// http://localhost:8080/mybatis07-spring-mvc/user/orders
@Controller
@RequestMapping("/user")
public class UserController {
    @Autowired
    UserMaper userMaper;

    @RequestMapping("/orders")
    public ModelAndView listall(HttpServletRequest request,HttpServ
        List<Order> orders=userMaper.getUserOrders(1);
        System.out.println("orders");
        ModelAndView mav=new ModelAndView("user_orders");
        mav.addObject("orders",orders);
        return mav;
    }
}
```

接下来还需要创建一个 web 页面作为结果输出，在目录 WebRoot/WEB-INF/pages 下创建一个名为 user_orders.jsp 文件，其代码如下：

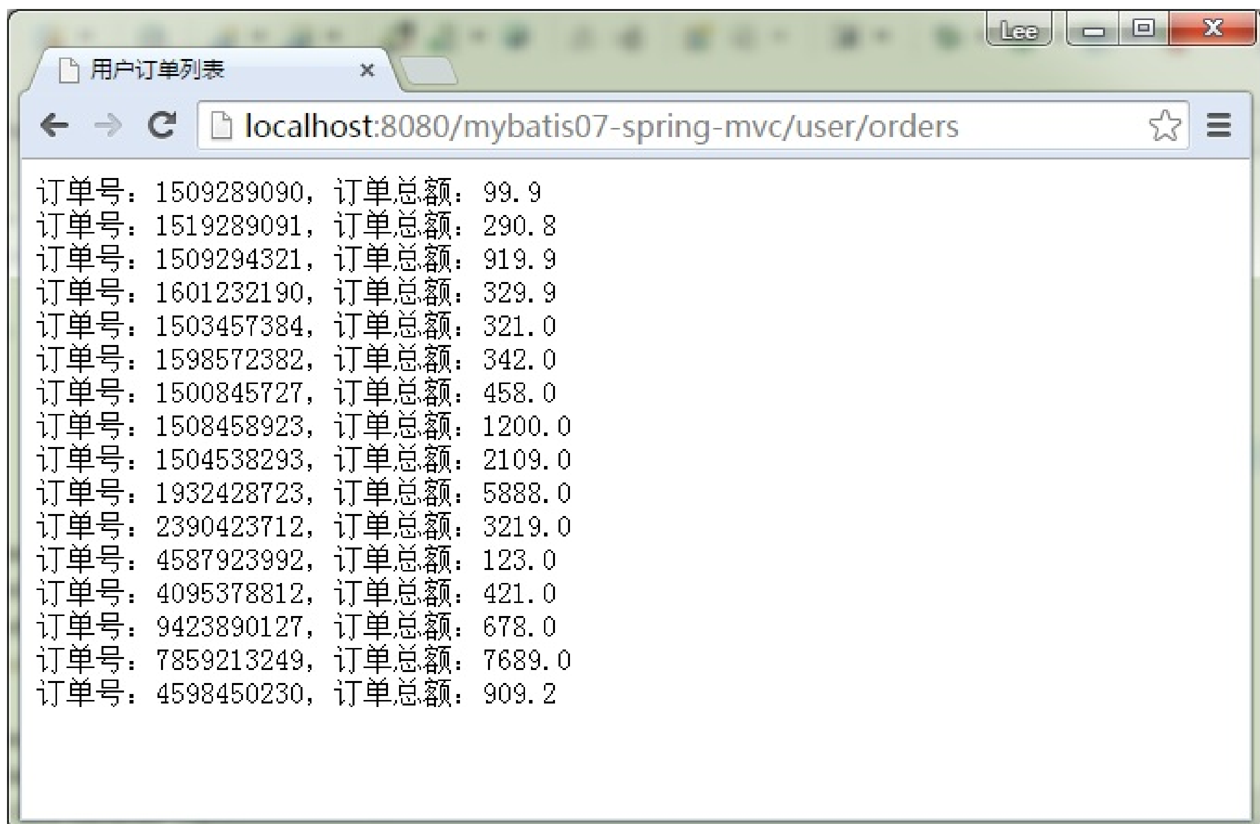
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "htt
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset
        <title>用户订单列表</title>
    </head>
    <body>
        <c:forEach items="${orders}" var="order">
        订单号：${order.orderNo }，订单总额：${order.money }<br />
        </c:forEach>
    </body>
</html>
```

然后启动 MyEclipse 中的 tomcat，具体如下图所示：



注：web.xml 还需要配置一下，详细见下载代码。

在浏览器中打开网址：http://localhost:8080/mybatis07-spring-mvc/user/orders，输出结果如下图所示：




工程代码下载：http://pan.baidu.com/s/1bnkE8cZ

Jar 包下载：http://pan.baidu.com/s/1bnyRJ9H

# MyBatis分页 - MyBatis教程

## 搞清楚什么是分页(pagination)

例如，在数据库的某个表里有1000条数据，我们每次只显示100条数据，在第1页显示第0到第99条，在第2页显示第100到199条，依次类推，这就是分页。

分页可以分为逻辑分页和物理分页。逻辑分页是我们的程序在显示每页的数据时，首先查询得到表中的1000条数据，然后成熟根据当前页的“页码”选出其中的100条数据来显示。

物理分页是程序先判断出该选出这1000条的第几条到第几条，然后数据库根据程序给出的信息查询出程序需要的100条返回给我们的程序。

## MyBatis 物理分页

MyBatis使用RowBounds实现的分页是逻辑分页,也就是先把数据记录全部查询出来,然在再根据 offset 和 limit 截断记录返回。

为了在数据库层面上实现物理分页，又不改变原来 MyBatis 的函数逻辑,可以编写 plugin 截获 MyBatis Executor 的 statementhandler，重写SQL来执行查询。

经常搭框架的人应该都清楚，框架搭建的核心就是配置文件。

在这里我们需要创建 web 工程。也需要用 mybatis与Spring mvc 集成起来，源码在本文结尾处下载，主要有以下几个方面的配置。

整个Mybatis分页示例要完成的步骤如下：

1、示例功能描述

2、创建工程

3、数据库表结构及数据记录

4、实例对象

5、配置文件

6、测试执行，输出结果

## 1、示例功能描述

在本示例中，需要使用 MyBatis和Spring MVC整合完成分页，完成这样的一个简单功能，即指定一个用户（ID=1），查询出这个用户关联的所有订单分页显示出来（使用的数据库是：MySQL）。

# 2、创建工程

首先创建一个工程的名称为：mybatis08-paging，在 src 源代码目录下建立文件夹 config，并将原来的 mybatis 配置文件 Configuration.xml 移动到这个文件夹中, 并在 config 文家夹中建立 Spring 配置文件：applicationContext.xml。工程结构目录如下：



# 3、数据库表结构及数据记录

在本示例中，用到两个表：用户表和订单表，其结构和数据记录如下：

```
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(64) NOT NULL DEFAULT '',
  `mobile` varchar(16) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');
INSERT INTO `user` VALUES ('2', 'saya', '13838009988');
```

订单表结构和数据如下：

```
CREATE TABLE `order` (
  `order_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_id` int(10) unsigned NOT NULL DEFAULT '0',
  `order_no` varchar(16) NOT NULL DEFAULT '',
  `money` float(10,2) unsigned DEFAULT '0.00',
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of order
-- ----------------------------
INSERT INTO `order` VALUES ('1', '1', '1509289090', '99.90');
INSERT INTO `order` VALUES ('2', '1', '1519289091', '290.80');
INSERT INTO `order` VALUES ('3', '1', '1509294321', '919.90');
INSERT INTO `order` VALUES ('4', '1', '1601232190', '329.90');
INSERT INTO `order` VALUES ('5', '1', '1503457384', '321.00');
INSERT INTO `order` VALUES ('6', '1', '1598572382', '342.00');
INSERT INTO `order` VALUES ('7', '1', '1500845727', '458.00');
INSERT INTO `order` VALUES ('8', '1', '1508458923', '1200.00');
INSERT INTO `order` VALUES ('9', '1', '1504538293', '2109.00');
INSERT INTO `order` VALUES ('10', '1', '1932428723', '5888.00');
INSERT INTO `order` VALUES ('11', '1', '2390423712', '3219.00');
INSERT INTO `order` VALUES ('12', '1', '4587923992', '123.00');
INSERT INTO `order` VALUES ('13', '1', '4095378812', '421.00');
INSERT INTO `order` VALUES ('14', '1', '9423890127', '678.00');
INSERT INTO `order` VALUES ('15', '1', '7859213249', '7689.00');
INSERT INTO `order` VALUES ('16', '1', '4598450230', '909.20');
```

## 4、实例对象

用户表和订单表分别对应两个实例对象，分别是：User.java 和 Order.java，它们都在 com.yiibai.pojo 包中。

User.java代码内容如下：

```
package com.yiibai.pojo;

import java.util.List;

/**
 * @describe: User
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class User {
    private int id;
    private String username;
    private String mobile;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getMobile() {
        return mobile;
    }
    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

}
```

Order.java代码内容如下：

```
package com.yiibai.pojo;

/**
 * @describe: User - 订单
 * @author: Yiibai
 * @version: V1.0
 * @copyright http://www.yiibai.com
 */
public class Order {
    private int orderId;
    private String orderNo;
    private float money;
    private int userId;
    private User user;

    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public int getOrderId() {
        return orderId;
    }
    public void setOrderId(int orderId) {
        this.orderId = orderId;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public String getOrderNo() {
        return orderNo;
    }
    public void setOrderNo(String orderNo) {
        this.orderNo = orderNo;
    }
    public float getMoney() {
        return money;
    }
    public void setMoney(float money) {
        this.money = money;
    }

}
```

# 5、配置文件

这个实例中有三个重要的配置文件，它们分别是：applicationContext.xml，Configuration.xml 以及 UserMaper.xml。

applicationContext.xml 配置文件里最主要的配置：

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:conte
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="
            http://www.springframework.org/schema/beans http://www.
            http://www.springframework.org/schema/aop http://www.sp
            http://www.springframework.org/schema/context http://ww
            http://www.springframework.org/schema/jee http://www.sp
            http://www.springframework.org/schema/tx http://www.spr
    default-autowire="byName" default-lazy-init="false">

    <!--本示例采用DBCP连接池，应预先把DBCP的jar包复制到工程的lib目录下。 --
    <context:property-placeholder location="classpath:/config/datab

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataS
        destroy-method="close" p:driverClassName="com.mysql.jdbc.Dr
        p:url="jdbc:mysql://127.0.0.1:3306/yiibai?characterEncoding
        p:username="root" p:password="" p:maxActive="10" p:maxIdle=
    </bean>

    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransa
        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessi
        <!--dataSource属性指定要用到的连接池-->
        <property name="dataSource" ref="dataSource" />
        <!--configLocation属性指定mybatis的核心配置文件-->
        <property name="configLocation" value="classpath:config/Con
        <!-- 所有配置的mapper文件 -->
        <property name="mapperLocations" value="classpath*:com/yiib
    </bean>

    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"
        <property name="basePackage" value="com.yiibai.maper" />
    </bean>
</beans>
```

配置文件 Configuration.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN'
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias alias="User" type="com.yiibai.pojo.User"/>
        <typeAlias alias="Order" type="com.yiibai.pojo.Order"/>
    </typeAliases>
    <!-- 与spring 集成之后,这些可以完全删除,数据库连接的管理交给 spring 去'
    <!--
    <environments default="development">
        <environment id="development">
        <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
            <property name="driver" value="com.mysql.jdbc.Driver"/>
            <property name="url" value="jdbc:mysql://127.0.0.1:330(
            <property name="username" value="root"/>
            <property name="password" value="password"/>
            </dataSource>
        </environment>
    </environments>
    -->

    <!-- 这里交给sqlSessionFactory 的 mapperLocations属性去得到所有配置
    <!--
    <mappers>
        <mapper resource="com/yihaomen/mapper/User.xml"/>
    </mappers>
    -->

</configuration>
```

UserMaper.xml 用于定义查询和数据对象映射，其内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.yiibai.maper.UserMaper">

        <!-- 为了返回list 类型而定义的returnMap -->
    <resultMap type="User" id="resultUser">
        <id column="id" property="id" />
        <result column="username" property="username" />
        <result column="mobile" property="mobile" />
    </resultMap>

    <!-- User 联合 Order 查询 方法的配置 (多对一的方式)  -->
    <resultMap id="resultUserOrders" type="Order">
        <id property="orderId" column="order_id" />
        <result property="orderNo" column="order_no" />
        <result property="money" column="money" />
        <result property="userId" column="user_id" />

        <association property="user" javaType="User">
            <id property="id" column="id" />
            <result property="username" column="username" />
            <result property="mobile" column="mobile" />
        </association>
    </resultMap>

    <select id="getUserOrders" parameterType="int" resultMap="resul
       SELECT u.*,o.* FROM `user` u, `order` o
              WHERE u.id=o.user_id AND u.id=#{id}
    </select>

    <select id="getUserById" resultMap="resultUser" parameterType='
        SELECT *
        FROM user
        WHERE id=#{id}
    </select>
</mapper>
```

# 6、测试执行，输出结果

我们创建一个控制器类在包 com.yiibai.controller 下，类的名称为：UserController.java，这里新增了一个方法：pageList， 对应请求URL是 /orderpages，其代码如下：

```
package com.yiibai.controller;

import java.util.List;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.yiibai.maper.UserMaper;
import com.yiibai.pojo.Order;
import com.yiibai.util.Page;

// http://localhost:8080/mybatis08-paging/user/orders
@Controller
@RequestMapping("/user")
public class UserController {
    @Autowired
    UserMaper userMaper;

    /**
     * 某一个用户下的所有订单
     *
     * @param request
     * @param response
     * @return
     */
    @RequestMapping("/orders")
    public ModelAndView listall(HttpServletRequest request,
            HttpServletResponse response) {
        List<Order> orders = userMaper.getUserOrders(1);
        System.out.println("orders");
        ModelAndView mav = new ModelAndView("user_orders");
        mav.addObject("orders", orders);
        return mav;
    }

    /**
     * 订单分页
     *
     * @param request
     * @param response
     * @return
     */
    @RequestMapping("/orderpages")
    public ModelAndView pageList(HttpServletRequest request,
            HttpServletResponse response) {
        int currentPage = request.getParameter("page") == null ? 1
                .parseInt(request.getParameter("page"));
        int pageSize = 3;
        if (currentPage <= 0) {
            currentPage = 1;
        }
        int currentResult = (currentPage - 1) * pageSize;
```

```
        System.out.println(request.getRequestURI());
        System.out.println(request.getQueryString());

        Page page = new Page();
        page.setShowCount(pageSize);
        page.setCurrentResult(currentResult);
        List<Order> orders = userMaper.getOrderListPage(page, 1);

        System.out.println("Current page =>" + page);

        int totalCount = page.getTotalResult();

        int lastPage = 0;
        if (totalCount % pageSize == 0) {
            lastPage = totalCount % pageSize;
        } else {
            lastPage = 1 + totalCount / pageSize;
        }

        if (currentPage >= lastPage) {
            currentPage = lastPage;
        }

        String pageStr = "";

        pageStr = String.format(
                "<a href=\"%s\">上一页</a>    <a href=\"%s\">下一页</
                        .getRequestURI()
                        + "?page=" + (currentPage - 1), request.get
                        + "?page=" + (currentPage + 1));

        // 制定视图，也就是list.jsp
        ModelAndView mav = new ModelAndView("pagelist");
        mav.addObject("orders", orders);
        mav.addObject("pagelist", pageStr);
        return mav;
    }
}
```

注意，在这个分页工程中，在 com.yiibai.util 包下新增了几个类，它们分别是：PagePlugin.java，Page.java, PageHelper.java，其中 PagePlugin 是针对 MyBatis 分页的插件。由于代码太多，这里列出 PagePlugin.java 的代码，其它两个类的代码有兴趣的读者可以在下载代码后阅读取研究。 PagePlugin.java 的代码如下所示：

```
package com.yiibai.util;

import java.lang.reflect.Field;
import java.sql.Connection;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import java.util.Properties;

import javax.xml.bind.PropertyException;

import org.apache.ibatis.executor.ErrorContext;
import org.apache.ibatis.executor.Executor;
import org.apache.ibatis.executor.ExecutorException;
import org.apache.ibatis.executor.statement.BaseStatementHandler;
import org.apache.ibatis.executor.statement.RoutingStatementHandler
import org.apache.ibatis.executor.statement.StatementHandler;
import org.apache.ibatis.mapping.BoundSql;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.mapping.ParameterMapping;
import org.apache.ibatis.mapping.ParameterMode;
import org.apache.ibatis.plugin.Interceptor;
import org.apache.ibatis.plugin.Intercepts;
import org.apache.ibatis.plugin.Invocation;
import org.apache.ibatis.plugin.Plugin;
import org.apache.ibatis.plugin.Signature;
import org.apache.ibatis.reflection.MetaObject;
import org.apache.ibatis.reflection.property.PropertyTokenizer;
import org.apache.ibatis.scripting.xmltags.ForEachSqlNode;
import org.apache.ibatis.session.Configuration;
import org.apache.ibatis.session.ResultHandler;
import org.apache.ibatis.session.RowBounds;
import org.apache.ibatis.type.TypeHandler;
import org.apache.ibatis.type.TypeHandlerRegistry;

@Intercepts( { @Signature(type = StatementHandler.class, method = '
public class PagePlugin implements Interceptor {

    private static String dialect = "";
    private static String pageSqlId = "";

    @SuppressWarnings("unchecked")
    public Object intercept(Invocation ivk) throws Throwable {

        if (ivk.getTarget() instanceof RoutingStatementHandler) {
            RoutingStatementHandler statementHandler = (RoutingStat
                    .getTarget();
            BaseStatementHandler delegate = (BaseStatementHandler)
                    .getValueByFieldName(statementHandler, "delegat
            MappedStatement mappedStatement = (MappedStatement) Ref
                    .getValueByFieldName(delegate, "mappedStatement

            if (mappedStatement.getId().matches(pageSqlId)) {
                BoundSql boundSql = delegate.getBoundSql();
                Object parameterObject = boundSql.getParameterObjec
                if (parameterObject == null) {
```

```
                throw new NullYiibaierException("parameterObjec
            } else {
                Connection connection = (Connection) ivk.getArg
                String sql = boundSql.getSql();
                String countSql = "select count(0) from (" + sq
                        + ") myCount";
                System.out.println("总数sql 语句:" + countSql);
                PreparedStatement countStmt = connection
                        .prepareStatement(countSql);
                BoundSql countBS = new BoundSql(mappedStatement
                        .getConfiguration(), countSql, boundSql
                        .getParameterMappings(), parameterObjec
                setParameters(countStmt, mappedStatement, count
                        parameterObject);
                ResultSet rs = countStmt.executeQuery();
                int count = 0;
                if (rs.next()) {
                    count = rs.getInt(1);
                }
                rs.close();
                countStmt.close();

                Page page = null;
                if (parameterObject instanceof Page) {
                    page = (Page) parameterObject;
                    page.setTotalResult(count);
                } else if (parameterObject instanceof Map) {
                    Map<String, Object> map = (Map<String, Obje
                    page = (Page) map.get("page");
                    if (page == null)
                        page = new Page();
                    page.setTotalResult(count);
                } else {
                    Field pageField = ReflectHelper.getFieldByF
                            parameterObject, "page");
                    if (pageField != null) {
                        page = (Page) ReflectHelper.getValueByF
                                parameterObject, "page");
                        if (page == null)
                            page = new Page();
                        page.setTotalResult(count);
                        ReflectHelper.setValueByFieldName(param
                                "page", page);
                    } else {
                        throw new NoSuchFieldException(paramete
                                .getClass().getName());
                    }
                }
                String pageSql = generatePageSql(sql, page);
                System.out.println("page sql:" + pageSql);
                ReflectHelper.setValueByFieldName(boundSql, "sq
            }
        }
```

```
        }
        return ivk.proceed();
    }

    private void setParameters(PreparedStatement ps,
            MappedStatement mappedStatement, BoundSql boundSql,
            Object parameterObject) throws SQLException {
        ErrorContext.instance().activity("setting parameters").obje
                mappedStatement.getParameterMap().getId());
        List<ParameterMapping> parameterMappings = boundSql
                .getParameterMappings();
        if (parameterMappings != null) {
            Configuration configuration = mappedStatement.getConfig
            TypeHandlerRegistry typeHandlerRegistry = configuration
                    .getTypeHandlerRegistry();
            MetaObject metaObject = parameterObject == null ? null
                    : configuration.newMetaObject(parameterObject);
            for (int i = 0; i < parameterMappings.size(); i++) {
                ParameterMapping parameterMapping = parameterMappin
                if (parameterMapping.getMode() != ParameterMode.OU
                    Object value;
                    String propertyName = parameterMapping.getPrope
                    PropertyTokenizer prop = new PropertyTokenizer(
                    if (parameterObject == null) {
                        value = null;
                    } else if (typeHandlerRegistry
                            .hasTypeHandler(parameterObject.getClas
                        value = parameterObject;
                    } else if (boundSql.hasAdditionalParameter(prop
                        value = boundSql.getAdditionalParameter(pro
                    } else if (propertyName
                            .startsWith(ForEachSqlNode.ITEM_PREFIX)
                            && boundSql.hasAdditionalParameter(prop
                        value = boundSql.getAdditionalParameter(pro
                        if (value != null) {
                            value = configuration.newMetaObject(val
                                    .getValue(
                                            propertyName.substring(
                                                    .getName().leng
                        }
                    } else {
                        value = metaObject == null ? null : metaObj
                                .getValue(propertyName);
                    }
                    TypeHandler typeHandler = parameterMapping.getT
                    if (typeHandler == null) {
                        throw new ExecutorException(
                                "There was no TypeHandler found for
                                        + propertyName + " of state
                                        + mappedStatement.getId());
                    }
                    typeHandler.setParameter(ps, i + 1, value, para
                            .getJdbcType());
```

```
                }
            }
        }
    }

    private String generatePageSql(String sql, Page page) {
        if (page != null && (dialect != null || !dialect.equals(""))
            StringBuffer pageSql = new StringBuffer();
            if ("mysql".equals(dialect)) {
                pageSql.append(sql);
                pageSql.append(" limit " + page.getCurrentResult()
                        + page.getShowCount());
            } else if ("oracle".equals(dialect)) {
                pageSql
                        .append("select * from (select tmp_tb.*,ROW
                pageSql.append(sql);
                pageSql.append(")  tmp_tb where ROWNUM<=");
                pageSql.append(page.getCurrentResult() + page.getSh
                pageSql.append(") where row_id>");
                pageSql.append(page.getCurrentResult());
            }
            return pageSql.toString();
        } else {
            return sql;
        }
    }

    public Object plugin(Object arg0) {
        // TODO Auto-generated method stub
        return Plugin.wrap(arg0, this);
    }

    public void setProperties(Properties p) {
        dialect = p.getProperty("dialect");
        if (dialect == null || dialect.equals("")) {
            try {
                throw new PropertyException("dialect property is no
            } catch (PropertyException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        pageSqlId = p.getProperty("pageSqlId");
        if (dialect == null || dialect.equals("")) {
            try {
                throw new PropertyException("pageSqlId property is
            } catch (PropertyException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

```
}
```

接下来部署 mybatis08-paging 这个工程，启动 tomcat，打开浏览器输入网址：http://localhost:8080/mybatis08-paging/user/orderpages，显示结果如下：

工程 mybatis08-paging 的代码下载：http://pan.baidu.com/s/1pJxhvNt

Jar 包下载：http://pan.baidu.com/s/1bnyRJ9H

# MyBatis动态SQL语句 - MyBatis教程

MyBatis 的强大特性之一便是它的动态 SQL。如果你有使用 JDBC 或其他类似框架的经验，你就能体会到根据不同条件拼接 SQL 语句有多么痛苦。拼接的时候要确保不能忘了必要的空格，还要注意省掉列名列表最后的逗号。利用动态 SQL 这一特性可以彻底摆脱这种痛苦。

通常使用动态 SQL 不可能是独立的一部分,MyBatis 当然使用一种强大的动态 SQL 语言来改进这种情形,这种语言可以被用在任意的 SQL 映射语句中。

动态 SQL 元素和使用 JSTL 或其他类似基于 XML 的文本处理器相似。在 MyBatis 之前的版本中,有很多的元素需要来了解。MyBatis 3 大大提升了它们,现在用不到原先一半的元素就可以了。MyBatis 采用功能强大的基于 OGNL 的表达式来消除其他元素。

mybatis 的动态sql语句是基于OGNL表达式的。可以方便的在 sql 语句中实现某些逻辑. 总体说来mybatis 动态SQL 语句主要有以下几类: 1. if 语句 (简单的条件判断) 2. choose (when,otherwize) ,相当于java 语言中的 switch ,与 jstl 中的choose 很类似. 3. trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀) 4. where (主要是用来简化sql语句中where条件判断的，能智能的处理 and or ,不必担心多余导致语法错误) 5. set (主要用于更新时) 6. foreach (在实现 mybatis in 语句查询时特别有用)

## if

动态 SQL 通常要做的事情是有条件地包含 where 子句的一部分。比如:

```
<select id="findActiveBlogWithTitleLike"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE state = ‘ACTIVE’
  <if test="title != null">
    AND title like #{title}
  </if>
</select>
```

这条语句提供了一个可选的文本查找类型的功能。如果没有传入“title”，那么所有处于“ACTIVE”状态的BLOG都会返回；反之若传入了“title”，那么就会把模糊查找“title”内容的BLOG结果返回(就这个例子而言，细心的读者会发现其中的参数值是可以包含一些掩码或通配符的)。

如果想可选地通过“title”和“author”两个条件搜索该怎么办呢？首先，改变语句的名称让它更具实际意义；然后只要加入另一个条件即可。

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = ‘ACTIVE’
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

## choose, when, otherwise

有些时候，我们不想用到所有的条件语句，而只想从中择其一二。针对这种情况，MyBatis 提供了 choose 元素，它有点像 Java 中的 switch 语句。

还是上面的例子，但是这次变为提供了“title”就按“title”查找，提供了“author”就按“author”查找，若两者都没有提供，就返回所有符合条件的BLOG(实际情况可能是由管理员按一定策略选出BLOG列表，而不是返回大量无意义的随机结果)。

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = ‘ACTIVE’
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and author.name != null">
      AND author_name like #{author.name}
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

## trim, where, set

前面几个例子已经合宜地解决了一个臭名昭著的动态 SQL 问题。现在考虑回到“if”示例，这次我们将“ACTIVE = 1”也设置成动态的条件，看看会发生什么。

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE
  <if test="state != null">
    state = #{state}
  </if>
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

如果这些条件没有一个能匹配上将会怎样？最终这条 SQL 会变成这样：

```
SELECT * FROM BLOG
WHERE
```

这会导致查询失败。如果仅仅第二个条件匹配又会怎样？这条 SQL 最终会是这样:

```
SELECT * FROM BLOG
WHERE
AND title like 'yiibai.com'
```

这个查询也会失败。这个问题不能简单的用条件句式来解决，如果你也曾经被迫这样写过，那么你很可能从此以后都不想再这样去写了。

MyBatis 有一个简单的处理，这在90%的情况下都会有用。而在不能使用的地方，你可以自定义处理方式来令其正常工作。一处简单的修改就能得到想要的效果：

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  <where>
    <if test="state != null">
         state = #{state}
    </if>
    <if test="title != null">
        AND title like #{title}
    </if>
    <if test="author != null and author.name != null">
        AND author_name like #{author.name}
    </if>
  </where>
</select>
```

where 元素知道只有在一个以上的if条件有值的情况下才去插入“WHERE”子句。而且，若最后的内容是“AND”或“OR”开头的，where 元素也知道如何将他们去除。

如果 where 元素没有按正常套路出牌，我们还是可以通过自定义 trim 元素来定制我们想要的功能。比如，和 where 元素等价的自定义 trim 元素为：

```
<trim prefix="WHERE" prefixOverrides="AND |OR "> ... </trim>
```

prefixOverrides 属性会忽略通过管道分隔的文本序列(注意此例中的空格也是必要的)。它带来的结果就是所有在 prefixOverrides 属性中指定的内容将被移除，并且插入 prefix 属性中指定的内容。

类似的用于动态更新语句的解决方案叫做 set。set 元素可以被用于动态包含需要更新的列，而舍去其他的。比如：

```
<update id="updateAuthorIfNecessary">
  update Author
    <set>
      <if test="username != null">username=#{username},</if>
      <if test="password != null">password=#{password},</if>
      <if test="email != null">email=#{email},</if>
      <if test="bio != null">bio=#{bio}</if>
    </set>
  where id=#{id}
</update>
```

这里，set 元素会动态前置 SET 关键字，同时也会消除无关的逗号，因为用了条件语句之后很可能就会在生成的赋值语句的后面留下这些逗号。

若你对等价的自定义 trim 元素的样子感兴趣，那这就应该是它的真面目：

```
<trim prefix="SET" suffixOverrides=","> ... </trim>
```

注意这里我们忽略的是后缀中的值，而又一次附加了前缀中的值。

## foreach

动态 SQL 的另外一个常用的必要操作是需要对一个集合进行遍历，通常是在构建 IN 条件语句的时候。比如：

```
<select id="selectPostIn" resultType="domain.blog.Post">
  SELECT *
  FROM POST P
  WHERE ID in
  <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
        #{item}
  </foreach>
</select>
```

foreach 元素的功能是非常强大的，它允许你指定一个集合，声明可以用在元素体内的集合项和索引变量。它也允许你指定开闭匹配的字符串以及在迭代中间放置分隔符。这个元素是很智能的，因此它不会偶然地附加多余的分隔符。

注意 你可以将一个 List 实例或者数组作为参数对象传给 MyBatis，当你这么做的时候，MyBatis 会自动将它包装在一个 Map 中并以名称为键。List 实例将会以“list”作为键，而数组实例的键将是“array”。

到此我们已经完成了涉及 XML 配置文件和 XML 映射文件的讨论。下一部分将详细探讨 Java API，这样才能从已创建的映射中获取最大利益。

## bind

bind 元素可以从 OGNL 表达式中创建一个变量并将其绑定到上下文。比如：

```
<select id="selectBlogsLike" resultType="Blog">
  <bind name="pattern" value="'%' + _parameter.getTitle() + '%'" /
  SELECT * FROM BLOG
  WHERE title LIKE #{pattern}
</select>
```

## Multi-db vendor support

一个配置了“_databaseId”变量的 databaseIdProvider 对于动态代码来说是可用的，这样就可以根据不同的数据库厂商构建特定的语句。比如下面的例子：

```
<insert id="insert">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    <if test="_databaseId == 'oracle'">
      select seq_users.nextval from dual
    </if>
    <if test="_databaseId == 'db2'">
      select nextval for seq_users from sysibm.sysdummy1"
    </if>
  </selectKey>
  insert into users values (#{id}, #{name})
</insert>
```

## 动态 SQL 中可插拔的脚本语言

MyBatis 从 3.2 开始支持可插拔的脚本语言，因此你可以在插入一种语言的驱动 (language driver)之后来写基于这种语言的动态 SQL 查询。

可以通过实现下面接口的方式来插入一种语言：

```
public interface LanguageDriver {
  ParameterHandler createParameterHandler(MappedStatement mappedSta
  SqlSource createSqlSource(Configuration configuration, XNode scri
  SqlSource createSqlSource(Configuration configuration, String scı
}
```

一旦有了自定义的语言驱动，你就可以在 mybatis-config.xml 文件中将它设置为默认语言：

```
<typeAliases>
  <typeAlias type="org.sample.MyLanguageDriver" alias="myLanguage"/
</typeAliases>
<settings>
  <setting name="defaultScriptingLanguage" value="myLanguage"/>
</settings>
```

除了设置默认语言，你也可以针对特殊的语句指定特定语言，这可以通过如下的 lang 属性来完成：

```
<select id="selectBlog" lang="myLanguage"> SELECT * FROM BLOG </se
```

或者在你正在使用的映射中加上注解 @Lang 来完成：

```
public interface Mapper {
  @Lang(MyLanguageDriver.class)
  @Select("SELECT * FROM BLOG")
  List<Blog> selectBlog();
}
```

# MyBatis SqlSessionDaoSupport实例 - MyBatis教程

在前面的章节中，我们已经讲到了基本的 mybatis 操作，但都是基于 mapper 隐射操作的，在 mybatis3 中这个 mapper 接口貌似充当了以前在ibatis 2中的 DAO 层的作用。但事实上，如果有这个mapper接口不能完成的工作，或者需要更复杂的扩展的时候，我们就需要自己写 DAO 层。 mybatis 3 也是支持 DAO 层设计的，类似于 ibatis2。下面我们结合一个实例来介绍。

首页我们创建一个工程为：mybatis11，再创建一个 com.yihaomen.dao 包，然后在里面分别创建接口UserDAO，并实现该接口 UserDAOImpl 。整个工程的目录结构如下：



UserDao接口的代码如下：

```
package com.yiibai.dao;

import java.util.List;

import com.yiibai.pojo.Order;

public interface UserDAO {
    public List<Order> getUserOrders(int userId);
}
```

UserDaoImpl 实现UserDao接口的代码如下：

```
package com.yiibai.dao;

import java.util.List;

import org.mybatis.spring.support.SqlSessionDaoSupport;
import org.springframework.stereotype.Repository;

import com.yiibai.pojo.Order;

@Repository
public class UserDAOImpl extends SqlSessionDaoSupport implements Us
    public List<Order> getUserOrders(int userId) {
        // TODO Auto-generated method stub
        return this.getSqlSession().selectList("com.yiibai.inter.IU
    }
}
```

控制类 UserController.java 的代码如下：

```
package com.yiibai.controller;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.yiibai.dao.UserDAO;
import com.yiibai.pojo.Order;
import com.yiibai.util.Page;

// http://localhost:8080/mybatis08-paging/user/orders
@Controller
@RequestMapping("/user")
public class UserController {
    //UserMaper userMaper;

    @Autowired
    UserDAO userDao;

    /**
     * 某一个用户下的所有订单（Dao方式）
     * URL => http://localhost:8080/mybatis11/user/lists
     *
     * @param request
     * @param response
     * @return
     */
    @RequestMapping("/lists")
    public ModelAndView listalldao(HttpServletRequest request,
            HttpServletResponse response) {
        List<Order> orders = this.userDao.getUserOrders(1);
        // 制定视图 =>list.jsp
        ModelAndView mav = new ModelAndView("lists");
        mav.addObject("orders", orders);
        return mav;
    }

}
```

下一步，我们运行检验结果（根据用户ID，读取这个用户的所有订单），打开浏览器，输入网址：http://localhost:8080/mybatis11/user/lists ，结果如下图所示：

MyBatis SqlSessionDao

localhost:8080/mybatis11/user/lists

| 编号 | 订单号 | 金额 |
|---|---|---|
| 1 | 1509289090 | 99.9 |
| 2 | 1519289091 | 290.8 |
| 3 | 1509294321 | 919.9 |
| 4 | 1601232190 | 329.9 |
| 5 | 1503457384 | 321.0 |
| 6 | 1598572382 | 342.0 |
| 7 | 1500845727 | 458.0 |
| 8 | 1508458923 | 1200.0 |
| 9 | 1504538293 | 2109.0 |
| 10 | 1932428723 | 5888.0 |
| 11 | 2390423712 | 3219.0 |
| 12 | 4587923992 | 123.0 |
| 13 | 4095378812 | 421.0 |
| 14 | 9423890127 | 678.0 |
| 15 | 7859213249 | 7689.0 |
| 16 | 4598450230 | 909.2 |

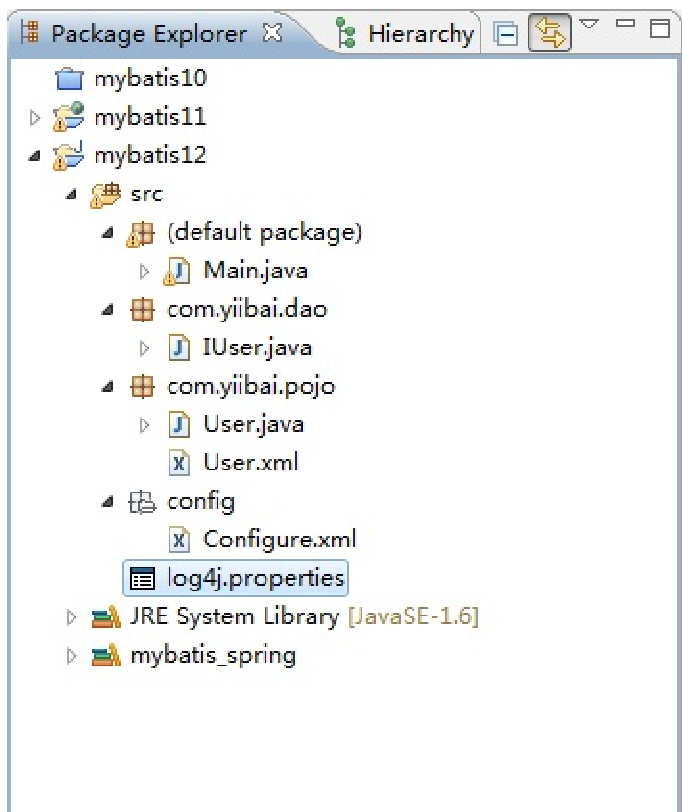
工程 mybatis11 的代码下载：http://pan.baidu.com/s/1sjrEL9V

Jar 包下载：http://pan.baidu.com/s/1bnyRJ9H

# MyBatis打印输出SQL语句 - MyBatis教程

Hibernate是可以配置 show_sql 显示 自动生成的SQL 语句，用 format_sql 可以格式化SQL 语句，但如果用 mybatis 怎么实现这个功能呢？如果你搜索看一下，基本都是通过配置日志来实现的，比如配置我们最常用的 log4j.properties 来实现。

首页我们创建一个 java 工程叫作：mybatis12，内容与之前 Mybatis+Spring 差不多，实现一个通过指定用户ID并读取其订单列表，来观察SQL的执行情况。其工程目录结构如下：




log4j.properties 内容如下：

```
# by yiibai.com
log4j.rootLogger=debug,stdout,logfile
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
#log4j.appender.stdout.Target=System.err
log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout
log4j.appender.logfile=org.apache.log4j.FileAppender
log4j.appender.logfile.File=C:/mybatis_show_sql.log
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm

log4j.logger.com.ibatis=DEBUG
log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG
log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG
log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEE
log4j.logger.java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
```

主类测试代码如下：

```
import java.io.Reader;
import java.text.MessageFormat;
import java.util.List;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.yiibai.dao.IUser;
import com.yiibai.pojo.User;

public class Main {
    private static SqlSessionFactory sqlSessionFactory;
    private static Reader reader;

    static {
        try {
            reader = Resources.getResourceAsReader("config/Configu
            sqlSessionFactory = new SqlSessionFactoryBuilder().buil
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static SqlSessionFactory getSession() {
        return sqlSessionFactory;
    }

    /**
```

```
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SqlSession session = sqlSessionFactory.openSession();
        try {
            //User user = (User) session.selectOne(
            //        "com.yiibai.mybatis.models.UserMapper.getUser
            IUser iuser = session.getMapper(IUser.class);
            getUserList();
            //testInsert();
            testUpdate();
            //testDelete();

        } finally {
            session.close();
        }
    }
    //
    public static void testInsert()
    {
        try
        {
            SqlSession session = sqlSessionFactory.openSession();
            IUser userMapper = session.getMapper(IUser.class);
            System.out.println("Test insert start...");
            User user = new User();
            user.setId(0);
            user.setName("Google");
            user.setDept("Tech");
            user.setWebsite("http://www.google.com");
            user.setPhone("120");
            userMapper.insertUser(user);
            session.commit();

            System.out.println("\r\nAfter insert");
            getUserList();
            System.out.println("Test insert finished...");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    // 用户列表
    public static void getUserList(){
        try
        {
            SqlSession session = sqlSessionFactory.openSession();
            IUser iuser = session.getMapper(IUser.class);
            System.out.println("Test Get start...");
            printUsers(iuser.getUserList());
```

```
            System.out.println("Test Get finished...");
        }catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    public static void testUpdate()
    {
        try
        {
            SqlSession session = sqlSessionFactory.openSession();
            IUser iuser = session.getMapper(IUser.class);
            System.out.println("Test update start...");
            printUsers(iuser.getUserList());
            User user = iuser.getUser(1);
            user.setName("New name");
            iuser.updateUser(user);
            session.commit();
            System.out.println("\r\nAfter update");
            printUsers(iuser.getUserList());
            System.out.println("Test update finished...");
        }catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public static void testDelete()
    {
        try
        {
            SqlSession session = sqlSessionFactory.openSession();
            IUser iuser = session.getMapper(IUser.class);
            System.out.println("Test delete start...");
            System.out.println("Before delete");
            printUsers(iuser.getUserList());
            iuser.deleteUser(3);
            session.commit();
            System.out.println("\r\nAfter delete");
            printUsers(iuser.getUserList());
            System.out.println("Test delete finished...");
        }catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    /**
     *
     *
     * @param users
     */
    private static void printUsers(final List<User> users)
    {
```

```
        int count = 0;

        for (User user : users)
        {
            System.out.println(MessageFormat.format("=============
            System.out.println("User Id: " + user.getId());
            System.out.println("User Name: " + user.getName());
            System.out.println("User Dept: " + user.getDept());
            System.out.println("User Website: " + user.getWebsite()
        }
    }
}
```

执行后，在MyEclise终端输出结果如下：

```
DEBUG - Logging initialized using 'class org.apache.ibatis.logging
DEBUG - PooledDataSource forcefully closed/removed all connections
DEBUG - PooledDataSource forcefully closed/removed all connections
DEBUG - PooledDataSource forcefully closed/removed all connections
DEBUG - PooledDataSource forcefully closed/removed all connections
Test Get start...
DEBUG - Opening JDBC Connection
DEBUG - Created connection 22927632.
DEBUG - Setting autocommit to false on JDBC Connection [com.mysql.j
DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@15dd91
DEBUG - ==>  Preparing: select * from user
DEBUG - ==> Parameters:
DEBUG - <==      Total: 2
============= User[1]=================
User Id: 1
User Name: New name
User Dept: Tech
User Website: http://www.yiibai.com
============= User[2]=================
User Id: 2
User Name: Hevi
User Dept: Tech
User Website: http://www.baidu.com
Test Get finished...
Test update start...
DEBUG - Opening JDBC Connection
DEBUG - Created connection 33189144.
DEBUG - Setting autocommit to false on JDBC Connection [com.mysql.j
DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d1
DEBUG - ==>  Preparing: select * from user
DEBUG - ==> Parameters:
DEBUG - <==      Total: 2
============= User[1]=================
User Id: 1
User Name: New name
```

```
User Dept: Tech
User Website: http://www.yiibai.com
============= User[2]=================
User Id: 2
User Name: Hevi
User Dept: Tech
User Website: http://www.baidu.com
DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d1
DEBUG - ==>  Preparing: select * from user where id=?
DEBUG - ==> Parameters: 1(Integer)
DEBUG - <==      Total: 1
DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d1
DEBUG - ==>  Preparing: UPDATE user SET name = ?, dept = ?, website
DEBUG - ==> Parameters: New name(String), Tech(String), http://www.
DEBUG - <==    Updates: 1
DEBUG - Committing JDBC Connection [com.mysql.jdbc.JDBC4Connection@

After update
DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d1
DEBUG - ==>  Preparing: select * from user
DEBUG - ==> Parameters:
DEBUG - <==      Total: 2
============= User[1]=================
User Id: 1
User Name: New name
User Dept: Tech
User Website: http://www.yiibai.com
============= User[2]=================
User Id: 2
User Name: Hevi
User Dept: Tech
User Website: http://www.baidu.com
Test update finished...
```

代码下载：http://pan.baidu.com/s/1jGk165o

Jar 包下载：http://pan.baidu.com/s/1bnyRJ9H

# Quartz教程

Quartz是一个开源的作业调度框架，可以让计划的程序任务一个预定义的日期和时间运行。Quartz可以用来创建简单或复杂的日程安排执行几十，几百，甚至是十万的作业数。

## Quartz是什么?

### 作业调度库

Quartz 是一种功能丰富的，开放源码的作业调度库，可以在几乎任何Java应用程序集成 - 从最小的独立的应用程序到规模最大电子商务系统。Quartz可以用来创建简单或复杂的日程安排执行几十，几百，甚至是十万的作业数 - 作业被定义为标准的Java组件，可以执行几乎任何东西，可以编程让它们执行。 Quartz调度包括许多企业级功能，如JTA事务和集群支持。

Quartz 是可自由使用，使用Apache 2.0 license授权方式。

## Quartz可以用来做什么?

如果应用程序需要在给定时间执行任务，或者如果系统有连续维护作业，那么Quartz是理想的解决方案。

使用Quartz作业调度应用的示例：

- 驱动处理工作流程：作为一个新的订单被初始化放置，调度作业到在正好两个小时内，它将检查订单的状态，如果订单确认消息尚未收到命令触发警告通知，以及改变订单的状态为“等待的干预”。
- 系统维护：调度工作给数据库的内容，每个工作日（节假日除外平日）在11:30 PM转储到一个XML文件中。
- 在应用程序内提供提醒服务。

# Quartz特点 - Quartz教程

## 运行环境

- Quartz 可以运行嵌入在另一个独立式应用程序
- Quartz 可以在应用程序服务器(或servlet容器)内被实例化，并且参与XA事务
- Quartz 可以作为一个独立的程序运行(其自己的Java虚拟机内)，可以通过RMI使用
- Quartz 可以被实例化，作为独立的项目集群(负载平衡和故障转移功能)，用于作业的执行

## 作业调度

作业被安排在一个给定的触发时运行。触发器可以使用以下指令的接近任何组合来创建：

- 在一天中的某个时间（到毫秒）
- 在一周的某几天
- 在每月的某一天
- 在一年中的某些日期
- 不在注册的日历中列出的特定日期（如商业节假日除外）
- 重复特定次数
- 重复进行，直到一个特定的时间/日期
- 无限重复
- 重复的延迟时间间隔

作业是由其创建者赋予的名字，也可以组织成命名组。触发器也可以给予名称和放置在组中，以方便地将它们调度内组织。作业可以被添加到所述调度器一次，而是具有多个触发器注册。在企业Java环境中，作业可以执行自己的工作作为分布式（XA）事务的一部分。

## 作业执行

- 作业可以实现简单的作业接口，为作业执行工作的任何Java类。
- Job类的实例可以通过Quartz被实例化，或者通过应用程序框架。
- 当触发时，调度通知实现JobListener和TriggerListener接口零个或多个Java对象（监听器可以是简单的Java对象，或EJB，JMS或发布者等）。这些监听器在作业已经执行之后通知。
- 由于作业完成后返回JobCompletionCode，它通知的成功或失败的调度。JobCompletionCode还可以指示的基础上，成功的话就采取行动调度/失败的代码 - 如立即重新执行作业。

## 作业持久性

- Quartz的设计包括可被实现以提供的作业存储各种机制一个作业存储接口
- 通过使用包含的JDBCJobStore，所有的作业和触发器配置为“非挥发性”都存储在通过JDBC关系数据库。
- 通过使用包含的RAMJobStore，所有的作业和触发器存储在RAM，因此不计划执行仍然存在 - 但这是无需使用外部数据库的优势。

## 事务

- 可以参与JTA事务，通过使用JobStoreCMT（JDBCJobStore的子类）。
- Quartz可以管理JTA事务（开始并提交它们）周围作业的执行，从而使作业执行的工作自动将JTA事务中发生。

## 集群

- 故障切换
- 负载均衡
- Quartz的内置的群集功能，通过JDBCJobStore（如上所述）依靠数据库持久
- Terracotta扩展Quartz提供集群功能，而不需要一个支持数据库

## 监听器和插件

- 应用程序可以捕捉事件的调度监控或通过实现一个或多个监听器接口控制工作/触发行为。
- 插件机制，可以用来添加功能，Quartz让作业执行过程中或工作负载和触发定义的历史不受限在一个文件中。
- 附带了一些“工厂建有”插件和监听器。

# Quartz2第一个程序 - Quartz教程

Quartz, 是一个企业级调度工作的框架，帮助Java应用程序到调度工作/任务在指定的日期和时间运行。

本教程教作为一个入门介绍如何开发使用调度工作（在写本教程时使用的最新Quartz 2.2.1 ）

## 1. 下载Quartz

可以从官方网站或Maven中央存储库下载Quartz库文件；

_File : _quartz.properties

```
org.quartz.scheduler.instanceName = MyScheduler
org.quartz.threadPool.threadCount = 3
org.quartz.jobStore.class =org.quartz.simpl.RAMJobStore
```

## 2. Quartz 作业

Quartz作业定义要运行什么？

*File : HelloJob*

```
package com.yiibai.common;

import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

public class HelloJob implements Job
{
    public void execute(JobExecutionContext context)
    throws JobExecutionException {

        System.out.println("Hello Quartz!");

    }
}
```

## 3. Quartz触发器

定义Quartz触发器，当Quartz运行在上面的Quartz作业。

像老版本的Quartz，仍然有两种类型的触发器在Quartz2，但API有些变化：

- SimpleTrigger – 允许设置开始时间，结束时间，重复间隔。
- CronTrigger – 允许UNIX cron表达式来指定日期和时间来运行作业。

SimpleTrigger – 每5秒运行。

```
    Trigger trigger = TriggerBuilder
    .newTrigger()
    .withIdentity("dummyTriggerName", "group1")
    .withSchedule(
        SimpleScheduleBuilder.simpleSchedule()
        .withIntervalInSeconds(5).repeatForever())
    .build();
```

CronTrigger – 每5秒运行。

```
    Trigger trigger = TriggerBuilder
    .newTrigger()
    .withIdentity("dummyTriggerName", "group1")
    .withSchedule(
        CronScheduleBuilder.cronSchedule("0/5 * * * * ?"))
    .build();
```

注意 请阅读 官方文档 更多的Quartz2触发器的例子。

## 4. Scheduler

调度类链接“工作”和“触发器”到一起，并执行它。

```
        Scheduler scheduler = new StdSchedulerFactory().getSchedule
        scheduler.start();
        scheduler.scheduleJob(job, trigger);
```

## 5. 完整的例子

Quartz2 两个 SimpleTrigger 和 CronTrigger 完整的例子。

SimpleTrigger的例子 - 每间隔5秒运行。

```
package com.yiibai.quartz;

import org.quartz.JobBuilder;
import org.quartz.JobDetail;
import org.quartz.Scheduler;
import org.quartz.SimpleScheduleBuilder;
import org.quartz.Trigger;
import org.quartz.TriggerBuilder;
import org.quartz.impl.StdSchedulerFactory;

public class SimpleTriggerExample {
    public static void main(String[] args) throws Exception {

        // Quartz 1.6.3
        // JobDetail job = new JobDetail();
        // job.setName("dummyJobName");
        // job.setJobClass(HelloJob.class);

        JobDetail job = JobBuilder.newJob(HelloJob.class)
            .withIdentity("dummyJobName", "group1").build();

                //Quartz 1.6.3
        // SimpleTrigger trigger = new SimpleTrigger();
        // trigger.setStartTime(new Date(System.currentTimeMillis()
        // trigger.setRepeatCount(SimpleTrigger.REPEAT_INDEFINITEL
        // trigger.setRepeatInterval(30000);

        // Trigger the job to run on the next round minute
        Trigger trigger = TriggerBuilder
            .newTrigger()
            .withIdentity("dummyTriggerName", "group1")
            .withSchedule(
                SimpleScheduleBuilder.simpleSchedule()
                    .withIntervalInSeconds(5).repeatForever())
            .build();

        // schedule it
        Scheduler scheduler = new StdSchedulerFactory().getSchedule
        scheduler.start();
        scheduler.scheduleJob(job, trigger);

    }
}
```

CronTrigger例子 - 同样，在每5秒运行作业。

```
package com.yiibai.quartz;

import org.quartz.CronScheduleBuilder;
import org.quartz.JobBuilder;
import org.quartz.JobDetail;
import org.quartz.Scheduler;
import org.quartz.Trigger;
import org.quartz.TriggerBuilder;
import org.quartz.impl.StdSchedulerFactory;

public class CronTriggerExample
{
    public static void main( String[] args ) throws Exception
    {
        //Quartz 1.6.3
        //JobDetail job = new JobDetail();
        //job.setName("dummyJobName");
        //job.setJobClass(HelloJob.class);
        JobDetail job = JobBuilder.newJob(HelloJob.class)
        .withIdentity("dummyJobName", "group1").build();

    //Quartz 1.6.3
        //CronTrigger trigger = new CronTrigger();
        //trigger.setName("dummyTriggerName");
        //trigger.setCronExpression("0/5 * * * * ?");

        Trigger trigger = TriggerBuilder
        .newTrigger()
        .withIdentity("dummyTriggerName", "group1")
        .withSchedule(
            CronScheduleBuilder.cronSchedule("0/5 * * * * ?"))
        .build();

        //schedule it
        Scheduler scheduler = new StdSchedulerFactory().getSchedule
        scheduler.start();
        scheduler.scheduleJob(job, trigger);

    }
}
```

代码下载：http://pan.baidu.com/s/1c0s6xoO

# Quartz2作业监听 - Quartz教程

在本教程中，我们将展示/介绍如何创建一个JobListener，跟踪运行工作状态在作业完成等。

P.S 这个例子是Quartz 2.1.5

## 1. Quartz 作业

作业 - 用于打印一个简单的信息，并抛出一个JobExecutionException进行测试。

*File : HelloJob.java*

```
package com.yiibai;

import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

public class HelloJob implements Job
{
    public void execute(JobExecutionContext context)
    throws JobExecutionException {

        System.out.println("Hello Quartz! - by yiibai.com");

        //Throw exception for testing
        throw new JobExecutionException("Testing Exception");
    }

}
```

## 2. JobListener

创建一个JobListener，只是实现了JobListener接口，并覆盖所有的接口的方法。

*File : HelloJobListener.java*

```
package com.yiibai.quartz.listener;

import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
import org.quartz.JobListener;

public class HelloJobListener implements JobListener {

    public static final String LISTENER_NAME = "dummyJobListenerNar

    @Override
    public String getName() {
        return LISTENER_NAME; //must return a name
    }

    // Run this if job is about to be executed.
    @Override
    public void jobToBeExecuted(JobExecutionContext context) {

        String jobName = context.getJobDetail().getKey().toString(
        System.out.println("jobToBeExecuted");
        System.out.println("Job : " + jobName + " is going to start

    }

    // No idea when will run this?
    @Override
    public void jobExecutionVetoed(JobExecutionContext context) {
        System.out.println("jobExecutionVetoed");
    }

    //Run this after job has been executed
    @Override
    public void jobWasExecuted(JobExecutionContext context,
            JobExecutionException jobException) {
        System.out.println("jobWasExecuted");

        String jobName = context.getJobDetail().getKey().toString(
        System.out.println("Job : " + jobName + " is finished...")

        if (!jobException.getMessage().equals("")) {
            System.out.println("Exception thrown by: " + jobName
                + " Exception: " + jobException.getMessage());
        }

    }

}
```

注意： 不知道什么是“jobExecutionVetoed”，并会在何时触发？

## 3. CronTrigger

例如上面HelloJobListener连接到调度和监控作业的状态。

*File : CronTriggerExample.java*

```
package com.yiibai.quartz;

import org.quartz.CronScheduleBuilder;
import org.quartz.JobBuilder;
import org.quartz.JobDetail;
import org.quartz.JobKey;
import org.quartz.Scheduler;
import org.quartz.Trigger;
import org.quartz.TriggerBuilder;
import org.quartz.impl.StdSchedulerFactory;
import org.quartz.impl.matchers.KeyMatcher;

import com.yiibai.quartz.listener.HelloJobListener;

public class CronTriggerExample {
    public static void main( String[] args ) throws Exception
    {

    JobKey jobKey = new JobKey("dummyJobName", "group1");
        JobDetail job = JobBuilder.newJob(HelloJob.class)
        .withIdentity(jobKey).build();

        Trigger trigger = TriggerBuilder
        .newTrigger()
        .withIdentity("dummyTriggerName", "group1")
        .withSchedule(
            CronScheduleBuilder.cronSchedule("0/5 * * * * ?"))
        .build();

        Scheduler scheduler = new StdSchedulerFactory().getSchedule

        //Listener attached to jobKey
        scheduler.getListenerManager().addJobListener(
            new HelloJobListener(), KeyMatcher.keyEquals(jobKey)
        );

        //Listener attached to group named "group 1" only.
        //scheduler.getListenerManager().addJobListener(
        //    new HelloJobListener(), GroupMatcher.jobGroupEquals('
        //);

        scheduler.start();
        scheduler.scheduleJob(job, trigger);

    }
}
```

运行CronTriggerExample.java, 这里是输出结果：

```
jobToBeExecuted
Job : group1.dummyJobName is going to start...
Hello Quartz! - by yiibai.com
jobWasExecuted
Job : group1.dummyJobName is started and finished...
Exception thrown by: group1.dummyJobName Exception: Testing Except:

jobToBeExecuted
Job : group1.dummyJobName is going to start...
Hello Quartz! - by yiibai.com
jobWasExecuted
Job : group1.dummyJobName is started and finished...
Exception thrown by: group1.dummyJobName Exception: Testing Except:
```

以下代码下载地址：http://pan.baidu.com/s/1pJ6xiwb

}

# Quartz执行多作业 - Quartz教程

在这个例子中，我们将介绍如何通过Quartz API 多个作业。在Quartz调度框架中，每个作业将被连接到一个唯一的触发，并且由调度器运行它。

P.S：在 Quartz 中，一个触发器触发多个作业是不可以的。

## 1. Quartz APIs

创建3个作业，JobA，JobB和JobC。

```
package com.yiibai.quartz;

import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

public class JobA implements Job {

    @Override
    public void execute(JobExecutionContext context)
        throws JobExecutionException {
        System.out.println("Job A is runing //every 5 seconds ");
    }

}
```

```
package com.yiibai.quartz;

import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

public class JobB implements Job {

    @Override
    public void execute(JobExecutionContext context)
        throws JobExecutionException {
        System.out.println("Job B is runing");
    }

}
```

```
package com.yiibai.quartz;

import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

public class JobC implements Job {

    @Override
    public void execute(JobExecutionContext context)
        throws JobExecutionException {
        System.out.println("Job C is runing");
    }

}
```

使用QuartzAPI声明上述3个作业，分配它们到特定触发器并调度它。

```
package com.yiibai.quartz;

import org.quartz.CronScheduleBuilder;
import org.quartz.JobBuilder;
import org.quartz.JobDetail;
import org.quartz.JobKey;
import org.quartz.Scheduler;
import org.quartz.Trigger;
import org.quartz.TriggerBuilder;
import org.quartz.impl.StdSchedulerFactory;

public class CronTriggerExample {
    public static void main( String[] args ) throws Exception
    {

    JobKey jobKeyA = new JobKey("jobA", "group1");
        JobDetail jobA = JobBuilder.newJob(JobA.class)
        .withIdentity(jobKeyA).build();

        JobKey jobKeyB = new JobKey("jobB", "group1");
        JobDetail jobB = JobBuilder.newJob(JobB.class)
        .withIdentity(jobKeyB).build();

        JobKey jobKeyC = new JobKey("jobC", "group1");
        JobDetail jobC = JobBuilder.newJob(JobC.class)
        .withIdentity(jobKeyC).build();

        Trigger trigger1 = TriggerBuilder
        .newTrigger()
        .withIdentity("dummyTriggerName1", "group1")
        .withSchedule(
            CronScheduleBuilder.cronSchedule("0/5 * * * * ?"))
        .build();
```

```
            Trigger trigger2 = TriggerBuilder
            .newTrigger()
            .withIdentity("dummyTriggerName2", "group1")
            .withSchedule(
                CronScheduleBuilder.cronSchedule("0/5 * * * * ?"))
            .build();

            Trigger trigger3 = TriggerBuilder
            .newTrigger()
            .withIdentity("dummyTriggerName3", "group1")
            .withSchedule(
                CronScheduleBuilder.cronSchedule("0/5 * * * * ?"))
            .build();

            Scheduler scheduler = new StdSchedulerFactory().getSchedule

            scheduler.start();
            scheduler.scheduleJob(jobA, trigger1);
            scheduler.scheduleJob(jobB, trigger2);
            scheduler.scheduleJob(jobC, trigger3);

        }
}
```

输出结果如上：

```
Job A is runing //every 5 seconds
Job B is runing
Job C is runing
Job A is runing //every 5 seconds
Job B is runing
Job C is runing
```

代码下载：http://pan.baidu.com/s/1qW4tkAw

# Quartz列出调度器所有作业 - Quartz教程

下面是两个代码片段展示如何列出所有Quartz的作业。Quartz2 API都发生了很大变化，所以语法和Quartz1.x是不同的

## 1. Quartz 2.2.1 示例

```
Scheduler scheduler = new StdSchedulerFactory().getScheduler();

   for (String groupName : scheduler.getJobGroupNames()) {

     for (JobKey jobKey : scheduler.getJobKeys(GroupMatcher.jobGrou

      String jobName = jobKey.getName();
      String jobGroup = jobKey.getGroup();

      //get job's trigger
      List<Trigger> triggers = (List<Trigger>) scheduler.getTrigger
      Date nextFireTime = triggers.get(0).getNextFireTime();

        System.out.println("[jobName] : " + jobName + " [groupName]
            + jobGroup + " - " + nextFireTime);

      }

    }
```

## 2. Quartz 1.8.6 示例

```
    Scheduler scheduler = new StdSchedulerFactory().getScheduler()

    //loop all group
    for (String groupName : scheduler.getJobGroupNames()) {

    //loop all jobs by groupname
    for (String jobName : scheduler.getJobNames(groupName)) {

          //get job's trigger
      Trigger[] triggers = scheduler.getTriggersOfJob(jobName,grou
      Date nextFireTime = triggers[0].getNextFireTime();

      System.out.println("[jobName] : " + jobName + " [groupName]
            + groupName + " - " + nextFireTime);

    }

    }
```

# Servlet 教程

# Servlet 简介

## Servlet 是什么？

Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。

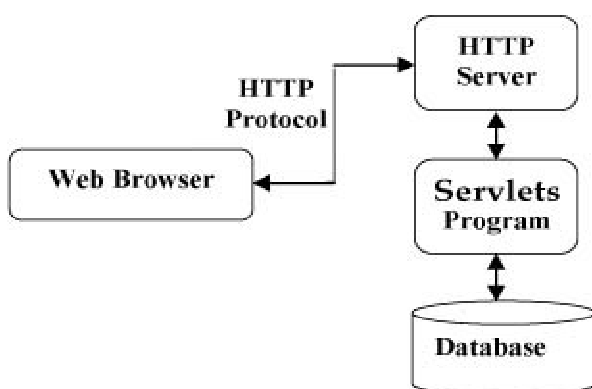
使用 Servlet，您可以收集来自网页表单的用户输入，呈现来自数据库或者其他源的记录，还可以动态创建网页。

Java Servlet 通常情况下与使用 CGI（Common Gateway Interface，公共网关接口）实现的程序可以达到异曲同工的效果。但是相比于 CGI，Servlet 有以下几点优势：

- 性能明显更好。
- Servlet 在 Web 服务器的地址空间内执行。这样它就没有必要再创建一个单独的进程来处理每个客户端请求。
- Servlet 是独立于平台的，因为它们是用 Java 编写的。
- 服务器上的 Java 安全管理器执行了一系列限制，以保护服务器计算机上的资源。因此，Servlet 是可信的。
- Java 类库的全部功能对 Servlet 来说都是可用的。它可以通过 sockets 和 RMI 机制与 applets、数据库或其他软件进行交互。

## Servlet 架构

下图显示了 Servlet 在 Web 应用程序中的位置。




## Servlet 任务

Servlet 执行以下主要任务：

- 读取客户端（浏览器）发送的显式的数据。这包括网页上的 HTML 表单，或者

也可以是来自 applet 或自定义的 HTTP 客户端程序的表单。
- 读取客户端（浏览器）发送的隐式的 HTTP 请求数据。这包括 cookies、媒体类型和浏览器能理解的压缩格式等等。
- 处理数据并生成结果。这个过程可能需要访问数据库，执行 RMI 或 CORBA 调用，调用 Web 服务，或者直接计算得出对应的响应。
- 发送显式的数据（即文档）到客户端（浏览器）。该文档的格式可以是多种多样的，包括文本文件（HTML 或 XML）、二进制文件（GIF 图像）、Excel 等。
- 发送隐式的 HTTP 响应到客户端（浏览器）。这包括告诉浏览器或其他客户端被返回的文档类型（例如 HTML），设置 cookies 和缓存参数，以及其他类似的任务。

# Servlet 包

Java Servlet 是运行在带有支持 Java Servlet 规范的解释器的 web 服务器上的 Java 类。

Servlet 可以使用 **javax.servlet** 和 **javax.servlet.http** 包创建，它是 Java 企业版的标准组成部分，Java 企业版是支持大型开发项目的 Java 类库的扩展版本。

这些类实现 Java Servlet 和 JSP 规范。在写本教程的时候，二者相应的版本分别是 Java Servlet 2.5 和 JSP 2.1。

Java Servlet 就像任何其他的 Java 类一样已经被创建和编译。在您安装 Servlet 包并把它们添加到您的计算机上的 Classpath 类路径中之后，您就可以通过 JDK 的 Java 编译器或任何其他编译器来编译 Servlet。

# 下一步呢？

接下来，本教程会带你一步一步地设置您的 Servlet 环境，以便开始后续的 Servlet 使用。因此，请系紧您的安全带，随我们一起开始 Servlet 的学习之旅吧！相信您会很喜欢这个教程的。

# Servlet 环境设置

开发环境是您可以开发、测试、运行 Servlet 的地方。

就像任何其他的 Java 程序，您需要通过使用 Java 编译器 **javac** 编译 Servlet，在编译 Servlet 应用程序后，将它部署在配置的环境中以便测试和运行。

这个开发环境设置包括以下步骤：

## 设置 Java 开发工具包（Java Development Kit）

这一步涉及到下载 Java 软件开发工具包（SDK，即 Software Development Kit），并适当地设置 PATH 环境变量。

您可以从 Oracle 的 Java 网站下载 SDK：Java SE Downloads。

一旦您下载了 SDK，请按照给定的指令来安装和配置设置。最后，设置 PATH 和 JAVA_HOME 环境变量指向包含 java 和 javac 的目录，通常分别为 java_install_dir/bin 和 java_install_dir。

如果您运行的是 Windows，并把 SDK 安装在 C:\jdk1.5.0_20 中，则需要在您的 C:\autoexec.bat 文件中放入下列的行：

```
set PATH=C:\jdk1.5.0_20\bin;%PATH%
set JAVA_HOME=C:\jdk1.5.0_20
```

或者，在 Windows NT/2000/XP 中，您也可以用鼠标右键单击"我的电脑"，选择"属性"，再选择"高级"，"环境变量"。然后，更新 PATH 的值，按下"确定"按钮。

在 Unix（Solaris、Linux 等）上，如果 SDK 安装在 /usr/local/jdk1.5.0_20 中，并且您使用的是 C shell，则需要在您的 .cshrc 文件中放入下列的行：

```
setenv PATH /usr/local/jdk1.5.0_20/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.5.0_20
```

另外，如果您使用集成开发环境（IDE，即 Integrated Development Environment），比如 Borland JBuilder、Eclipse、IntelliJ IDEA 或 Sun ONE Studio，编译并运行一个简单的程序，以确认该 IDE 知道您安装的 Java 路径。

## 设置 Web 服务器：Tomcat

在市场上有许多 Web 服务器支持 Servlet。有些 Web 服务器是免费下载的，Tomcat 就是其中的一个。

Apache Tomcat 是一款 Java Servlet 和 JavaServer Pages 技术的开源软件实现，可以作为测试 Servlet 的独立服务器，而且可以集成到 Apache Web 服务器。下面是在电脑上安装 Tomcat 的步骤：

- 从 http://tomcat.apache.org/ 上下载最新版本的 Tomcat。
- 一旦您下载了 Tomcat，解压缩到一个方便的位置。例如，如果您使用的是 Windows，则解压缩到 C:\apache-tomcat-5.5.29 中，如果您使用的是 Linux/Unix，则解压缩到 /usr/local/apache-tomcat-5.5.29 中，并创建 CATALINA_HOME 环境变量指向这些位置。

在 Windows 上，可以通过执行下面的命令来启动 Tomcat：

```
%CATALINA_HOME%\bin\startup.bat

or

C:\apache-tomcat-5.5.29\bin\startup.bat
```

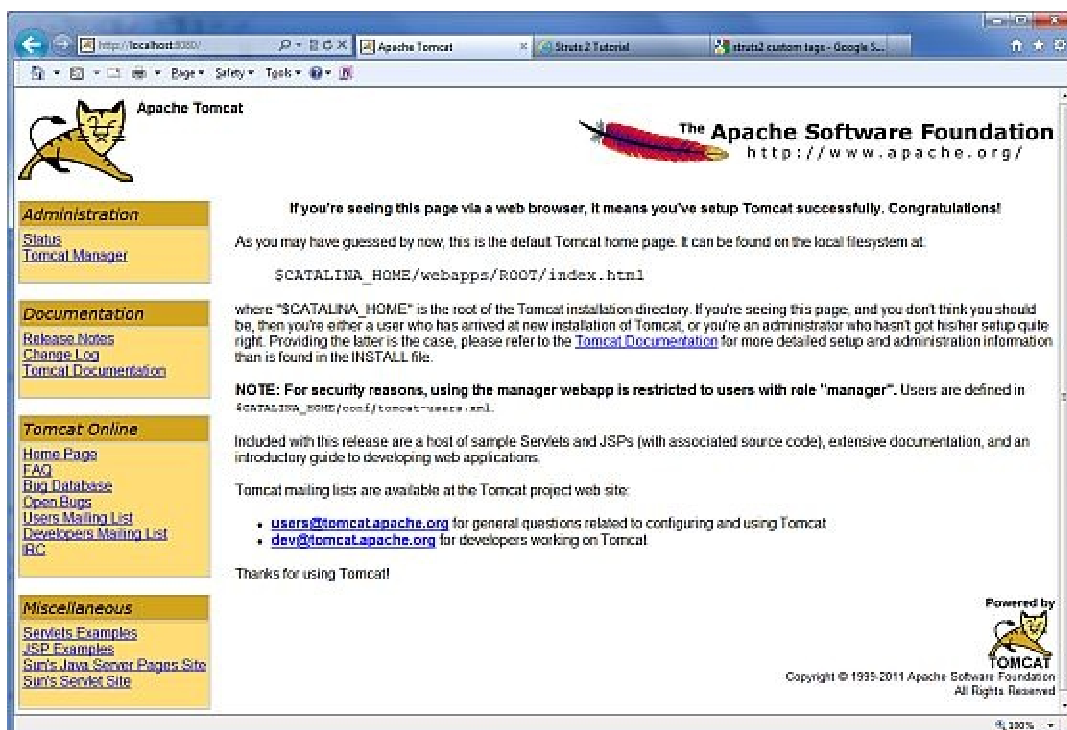
在 Unix（Solaris、Linux 等） 上，可以通过执行下面的命令来启动 Tomcat：

```
$CATALINA_HOME/bin/startup.sh

or

/usr/local/apache-tomcat-5.5.29/bin/startup.sh
```

Tomcat 启动后，可以通过在浏览器地址栏输入 **http://localhost:8080/** 访问 Tomcat 中的默认应用程序。如果一切顺利，那么会显示以下结果：



有关配置和运行 Tomcat 的进一步信息可以查阅应用程序安装的文档，或者可以访问 Tomcat 网站：http://tomcat.apache.org。

在 Windows 上，可以通过执行下面的命令来停止 Tomcat：

```
C:\apache-tomcat-5.5.29\bin\shutdown
```

在 Unix（Solaris、Linux 等） 上，可以通过执行下面的命令来停止 Tomcat：

```
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

# 设置 CLASSPATH

由于 Servlet 不是 Java 平台标准版的组成部分，所以您必须为编译器指定 Servlet 类的路径。

如果您运行的是 Windows，则需要在您的 C:\autoexec.bat 文件中放入下列的行：

```
set CATALINA=C:\apache-tomcat-5.5.29
set CLASSPATH=%CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%
```

或者，在 Windows NT/2000/XP 中，您也可以用鼠标右键单击"我的电脑"，选择"属性"，再选择"高级"，"环境变量"。然后，更新 CLASSPATH 的值，按下"确定"按钮。

在 Unix（Solaris、Linux 等）上，如果您使用的是 C shell，则需要在您的 .cshrc 文件中放入下列的行：

```
setenv CATALINA=/usr/local/apache-tomcat-5.5.29
setenv CLASSPATH $CATALINA/common/lib/servlet-api.jar:$CLASSPATH
```

注意：假设您的开发目录是 C:\ServletDevel（在 Windows 上）或 /user/ServletDevel（在 UNIX 上），那么您还需要在 CLASSPATH 中添加这些目录，添加方式与上面的添加方式类似。

# Servlet 生命周期

Servlet 生命周期可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：

- Servlet 通过调用 **init ()** 方法进行初始化。
- Servlet 调用 **service()** 方法来处理客户端的请求。
- Servlet 通过调用 **destroy()** 方法终止（结束）。
- 最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

现在让我们详细讨论生命周期的方法。

## init() 方法

init 方法被设计成只调用一次。它在第一次创建 Servlet 时被调用，在后续每次用户请求时不再调用。因此，它是用于一次性初始化，就像 Applet 的 init 方法一样。

Servlet 创建于用户第一次调用对应于该 Servlet 的 URL 时，但是您也可以指定 Servlet 在服务器第一次启动时被加载。

当用户调用一个 Servlet 时，就会创建一个 Servlet 实例，每一个用户请求都会产生一个新的线程，适当的时候移交给 doGet 或 doPost 方法。init() 方法简单地创建或加载一些数据，这些数据将被用于 Servlet 的整个生命周期。

init 方法的定义如下：

```
public void init() throws ServletException {
  // 初始化代码...
}
```

## service() 方法

service() 方法是执行实际任务的主要方法。Servlet 容器（即 Web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并把格式化的响应写回给客户端。

每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。

下面是该方法的特征：

```
public void service(ServletRequest request,
                    ServletResponse response)
      throws ServletException, IOException{
}
```

service() 方法由容器调用，service 方法在适当的时候调用 doGet、doPost、doPut、doDelete 等方法。所以，您不用对 service() 方法做任何动作，您只需要根据来自客户端的请求类型来重载 doGet() 或 doPost() 即可。

doGet() 和 doPost() 方法是每次服务请求中最常用的方法。下面是这两种方法的特征。

# doGet() 方法

GET 请求来自于一个 URL 的正常请求，或者来自于一个未指定 METHOD 的 HTML 表单，它由 doGet() 方法处理。

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet 代码
}
```

# doPost() 方法

POST 请求来自于一个特别指定了 METHOD 为 POST 的 HTML 表单，它由 doPost() 方法处理。

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet 代码
}
```

# destroy() 方法

destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。destroy() 方法可以让您的 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。

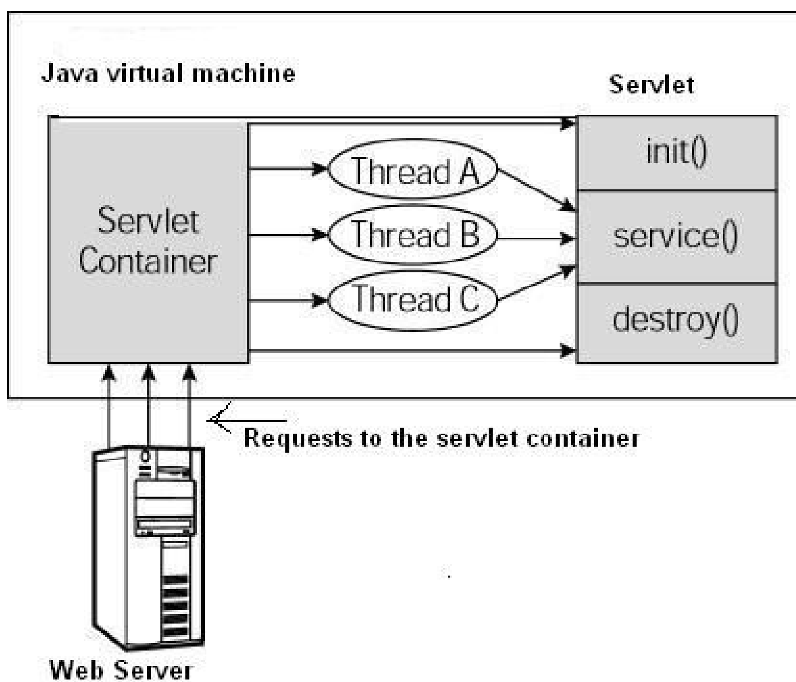
在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收。destroy 方法定义如下所示：

```
public void destroy() {
  // 终止化代码...
}
```

# 架构图

下图显示了一个典型的 Servlet 生命周期方案。

- 第一个到达服务器的 HTTP 请求被委派到 Servlet 容器。
- Servlet 容器在调用 service() 方法之前加载 Servlet。
- 然后 Servlet 容器处理由多个线程产生的多个请求，每个线程执行一个单一的 Servlet 实例的 service() 方法。



# Servlet 实例

Servlet 是服务 HTTP 请求并实现 **javax.servlet.Servlet** 接口的 Java 类。Web 应用程序开发人员通常编写 Servlet 来扩展 javax.servlet.http.HttpServlet，并实现 Servlet 接口的抽象类专门用来处理 HTTP 请求。

## Hello World 示例代码

下面是 Servlet 输出 Hello World 的示例源代码：

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException
  {
      // 执行必需的初始化
      message = "Hello World";
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      // 实际的逻辑是在这里
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
  }

  public void destroy()
  {
      // 什么也不做
  }
}
```

## 编译 Servlet

让我们把上面的代码写在 HelloWorld.java 文件中，把这个文件放在 C:\ServletDevel（在 Windows 上）或 /usr/ServletDevel（在 UNIX 上）中，您还需要把这些目录添加到 CLASSPATH 中。

假设您的环境已经正确地设置，进入 **ServletDevel** 目录，并编译 HelloWorld.java，如下所示：

```
$ javac HelloWorld.java
```

如果 Servlet 依赖于任何其他库，您必须在 CLASSPATH 中包含那些 JAR 文件。在这里，我只包含了 servlet-api.jar JAR 文件，因为我没有在 Hello World 程序中使用任何其他库。

该命令行使用 Sun Microsystems Java 软件开发工具包（JDK）内置的 javac 编译器。为使该命令正常工作，您必须 PATH 环境变量中使用的 Java SDK 的位置。

如果一切顺利，上面编译会在同一目录下生成 HelloWorld.class 文件。下一节将讲解已编译的 Servlet 如何部署在生产中。

## Servlet 部署

默认情况下，Servlet 应用程序位于路径 <Tomcat-installation-directory>/webapps/ROOT 下，且类文件放在 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中。

如果您有一个完全合格的类名称 **com.myorg.MyServlet**，那么这个 Servlet 类必须位于 WEB-INF/classes/com/myorg/MyServlet.class 中。

现在，让我们把 HelloWorld.class 复制到 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中，并在位于 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/ 的 **web.xml** 文件中创建以下条目：

```
    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorld</servlet-name>
        <url-pattern>/HelloWorld</url-pattern>
    </servlet-mapping>
```

上面的条目要被创建在 web.xml 文件中的 <web-app>...</web-app> 标签内。在该文件中可能已经有各种可用的条目，但不要在意。

到这里，您基本上已经完成了，现在让我们使用 <Tomcat-installation-directory>\bin\startup.bat（在 Windows 上）或 <Tomcat-installation-directory>/bin/startup.sh（在 Linux/Solaris 等上）启动 tomcat 服务器，最后在浏览器的地址栏中输入 **http://localhost:8080/HelloWorld**。如果一切顺利，您会看到下面的结果：



# Servlet 表单数据

很多情况下，需要传递一些信息，从浏览器到 Web 服务器，最终到后台程序。浏览器使用两种方法可将这些信息传递到 Web 服务器，分别为 GET 方法和 POST 方法。

## GET 方法

GET 方法向页面请求发送已编码的用户信息。页面和已编码的信息中间用 ? 字符分隔，如下所示：

```
http://www.test.com/hello?key1=value1&key2=value2
```

GET 方法是默认的从浏览器向 Web 服务器传递信息的方法，它会产生一个很长的字符串，出现在浏览器的地址栏中。如果您要向服务器传递的是密码或其他的敏感信息，请不要使用 GET 方法。GET 方法有大小限制：请求字符串中最多只能有 1024 个字符。

这些信息使用 QUERY_STRING 头传递，并可以通过 QUERY_STRING 环境变量访问，Servlet 使用 **doGet()** 方法处理这种类型的请求。

## POST 方法

另一个向后台程序传递信息的比较可靠的方法是 POST 方法。POST 方法打包信息的方式与 GET 方法基本相同，但是 POST 方法不是把信息作为 URL 中 ? 字符后的文本字符串进行发送，而是把这些信息作为一个单独的消息。消息以标准输出的形式传到后台程序，您可以解析和使用这些标准输出。Servlet 使用 doPost() 方法处理这种类型的请求。

## 使用 Servlet 读取表单数据

Servlet 处理表单数据，这些数据会根据不同的情况使用不同的方法自动解析：

- **getParameter()**：您可以调用 request.getParameter() 方法来获取表单参数的值。
- **getParameterValues()**：如果参数出现一次以上，则调用该方法，并返回多个值，例如复选框。
- **getParameterNames()**：如果您想要得到当前请求中的所有参数的完整列表，则调用该方法。

## 使用 URL 的 GET 方法实例

下面是一个简单的 URL，将使用 GET 方法向 HelloForm 程序传递两个值。

**http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI**

下面是处理 Web 浏览器输入的 **HelloForm.java** Servlet 程序。我们将使用 **getParameter()** 方法，可以很容易地访问传递的信息：

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "使用 GET 方法读取表单数据";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<ul>\n" +
                "  <li><b>名字</b>："
                + request.getParameter("first_name") + "\n" +
                "  <li><b>姓氏</b>："
                + request.getParameter("last_name") + "\n" +
                "</ul>\n" +
                "</body></html>");
  }
}
```

假设您的环境已经正确地设置，编译 HelloForm.java，如下所示：

```
$ javac HelloForm.java
```

如果一切顺利，上述编译会产生 **HelloForm.class** 文件。接下来，您就必须把该类文件复制到 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中，并在位于 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/ 的

**web.xml** 文件中创建以下条目：

```
    <servlet>
        <servlet-name>HelloForm</servlet-name>
        <servlet-class>HelloForm</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloForm</servlet-name>
        <url-pattern>/HelloForm</url-pattern>
    </servlet-mapping>
```

现在在浏览器的地址栏中输入 *http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI*，并在触发上述命令之前确保已经启动 Tomcat 服务器。如果一切顺利，您会得到下面的结果：

```
<h1>使用 GET 方法读取表单数据</h1>

<ul>
<li><b>名字<b>：ZARA</li>
<li><b>姓氏<b>：ALI</li>
</ul>
```

# 使用表单的 GET 方法实例

下面是一个简单的实例，使用 HTML 表单和提交按钮传递两个值。我们将使用相同的 Servlet HelloForm 来处理输入。

```
<html>
<body>
<form action="HelloForm" method="GET">
名字：<input type="text" name="first_name">
<br />
姓氏：<input type="text" name="last_name" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

保存这个 HTML 到 hello.htm 文件中，并把它放在 <Tomcat-installation-directory>/webapps/ROOT 目录下。当您访问 *http://localhost:8080/Hello.htm* 时，下面是上面表单的实际输出。

```
<form action="javascript:void();" method="get" target="_blank">名字
姓氏：<input type="text" name="last_name"> <input type="button" valu
```

尝试输入名字和姓氏，然后点击"提交"按钮，在您本机上查看输出结果。基于所提供的输入，它会产生与上一个实例类似的结果。

## 使用表单的 **POST** 方法实例

让我们对上面的 Servlet 做小小的修改，以便它可以处理 GET 和 POST 方法。下面的 **HelloForm.java** Servlet 程序使用 GET 和 POST 方法处理由 Web 浏览器给出的输入。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class HelloForm extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

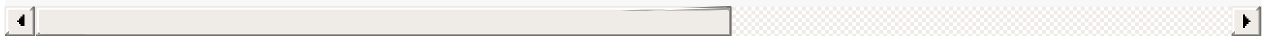
      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<ul>\n" +
                "  <li><b>名字</b>："
                + request.getParameter("first_name") + "\n" +
                "  <li><b>姓氏</b>："
                + request.getParameter("last_name") + "\n" +
                "</ul>\n" +
                "</body></html>");
  }
  // 处理 POST 方法请求的方法
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
  }
}
```

现在，编译部署上述的 Servlet，并使用带有 POST 方法的 Hello.htm 进行测试，如下所示：

```
<html>
<body>
<form action="HelloForm" method="POST">
名字：<input type="text" name="first_name">
<br />
姓氏：<input type="text" name="last_name" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

下面是上面表单的实际输出，尝试输入名字和姓氏，然后点击"提交"按钮，在您本机上查看输出结果。

```
<form action="javascript:void();" method="get" target="_blank">名字
姓氏：<input type="text" name="last_name"> <input type="button" valu
```



基于所提供的输入，它会产生与上一个实例类似的结果。

## 将复选框数据传递到 Servlet 程序

当需要选择一个以上的选项时，则使用复选框。

下面是一个 HTML 代码实例 CheckBox.htm，一个带有两个复选框的表单。

```
<html>
<body>
<form action="CheckBox" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> 数学
<input type="checkbox" name="physics"  /> 物理
<input type="checkbox" name="chemistry" checked="checked" />
                                                化学
<input type="submit" value="选择学科" />
</form>
</body>
</html>
```

这段代码的结果是下面的表单：

```
<form action="javascript:void();" method="get" target="_blank"><inp
```



下面是 CheckBox.java Servlet 程序，处理 Web 浏览器给出的复选框输入。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class CheckBox extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "读取复选框数据";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<ul>\n" +
                "  <li><b>数学标识：</b>: "
                + request.getParameter("maths") + "\n" +
                "  <li><b>物理标识：</b>: "
                + request.getParameter("physics") + "\n" +
                "  <li><b>化学标识：</b>: "
                + request.getParameter("chemistry") + "\n" +
                "</ul>\n" +
                "</body></html>");
  }
  // 处理 POST 方法请求的方法
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
  }
}
```

上面的实例将显示下面的结果：

```
<h1>读取复选框数据</h1>

<ul>
<li><b>数学标识：</b>on</li>
<li><b>物理标识：</b>null</li>
<li><b>化学标识：</b>on</li>
</ul>
```

# 读取所有的表单参数

以下是通用的实例，使用 HttpServletRequest 的 **getParameterNames()** 方法读取所有可用的表单参数。该方法返回一个枚举，其中包含未指定顺序的参数名。

一旦我们有一个枚举，我们可以以标准方式循环枚举，使用 *hasMoreElements()* 方法来确定何时停止，使用 *nextElement()* 方法来获取每个参数的名称。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 扩展 HttpServlet 类
public class ReadParams extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "读取所有的表单数据";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
        "<tr bgcolor=\"#949494\">\n" +
        "<th>参数名称</th><th>参数值</th>\n"+
        "</tr>\n");

      Enumeration paramNames = request.getParameterNames();
```

```
      while(paramNames.hasMoreElements()) {
         String paramName = (String)paramNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n<td>");
         String[] paramValues =
                request.getParameterValues(paramName);
         // 读取单个值的数据
         if (paramValues.length == 1) {
           String paramValue = paramValues[0];
           if (paramValue.length() == 0)
             out.println("<i>No Value</i>");
           else
             out.println(paramValue);
         } else {
             // 读取多个值的数据
             out.println("<ul>");
             for(int i=0; i < paramValues.length; i++) {
                out.println("<li>" + paramValues[i]);
             }
             out.println("</ul>");
         }
      }
      out.println("</tr>\n</table>\n</body></html>");
   }
   // 处理 POST 方法请求的方法
   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
   }
}
```

现在，通过下面的表单尝试上面的 Servlet：

```
<html>
<body>
<form action="ReadParams" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> 数学
<input type="checkbox" name="physics"  /> 物理
<input type="checkbox" name="chemistry" checked="checked" /> 化学
<input type="submit" value="选择学科" />
</form>
</body>
</html>
```

现在使用上面的表单调用 Servlet，将产生以下结果：

# 读取所有的表单数据

| 参数名称 | 参数值 |
| --- | --- |
| maths | on |
| chemistry | on |

您可以尝试使用上面的 Servlet 来读取其他的表单数据，比如文本框、单选按钮或下拉框等。

# Servlet 客户端 HTTP 请求

当浏览器请求网页时，它会向 Web 服务器发送特定信息，这些信息不能被直接读取，因为这些信息是作为 HTTP 请求的头的一部分进行传输的。您可以查看 HTTP 协议 了解更多相关信息。

以下是来自于浏览器端的重要头信息，您可以在 Web 编程中频繁使用：

| 头信息 | 描述 |
|---|---|
| Accept | 这个头信息指定浏览器或其他客户端可以处理的 MIME 类型。值 **image/png** 或 **image/jpeg** 是最常见的两种可能值。 |
| Accept-Charset | 这个头信息指定浏览器可以用来显示信息的字符集。例如 ISO-8859-1。 |
| Accept-Encoding | 这个头信息指定浏览器知道如何处理的编码类型。值 **gzip** 或 **compress** 是最常见的两种可能值。 |
| Accept-Language | 这个头信息指定客户端的首选语言，在这种情况下，Servlet 会产生多种语言的结果。例如，en、en-us、ru 等。 |
| Authorization | 这个头信息用于客户端在访问受密码保护的网页时识别自己的身份。 |
| Connection | 这个头信息指示客户端是否可以处理持久 HTTP 连接。持久连接允许客户端或其他浏览器通过单个请求来检索多个文件。值 **Keep-Alive** 意味着使用了持续连接。 |
| Content-Length | 这个头信息只适用于 POST 请求，并给出 POST 数据的大小（以字节为单位）。 |
| Cookie | 这个头信息把之前发送到浏览器的 cookies 返回到服务器。 |
| Host | 这个头信息指定原始的 URL 中的主机和端口。 |
| If-Modified-Since | 这个头信息表示只有当页面在指定的日期后已更改时，客户端想要的页面。如果没有新的结果可以使用，服务器会发送一个 304 代码，表示 **Not Modified** 头信息。 |
| If-Unmodified-Since | 这个头信息是 If-Modified-Since 的对立面，它指定只有当文档早于指定日期时，操作才会成功。 |
| Referer | 这个头信息指示所指向的 Web 页的 URL。例如，如果您在网页 1，点击一个链接到网页 2，当浏览器请求网页 2 时，网页 1 的 URL 就会包含在 Referer 头信息中。 |
| User-Agent | 这个头信息识别发出请求的浏览器或其他客户端，并可以向不同类型的浏览器返回不同的内容。 |

# 读取 HTTP 头的方法

下面的方法可用在 Servlet 程序中读取 HTTP 头。这些方法通过 *HttpServletRequest* 对象可用。

| 方法 | 描述 |
|---|---|
| **Cookie[] getCookies()** | 返回一个数组，包含客户端发送该请求的所有的 Cookie 对象。 |
| **Enumeration getAttributeNames()** | 返回一个枚举，包含提供给该请求可用的属性名称。 |
| **Enumeration getHeaderNames()** | 返回一个枚举，包含在该请求中包含的所有的头名。 |
| **Enumeration getParameterNames()** | 返回一个 String 对象的枚举，包含在该请求中包含的参数的名称。 |
| **HttpSession getSession()** | 返回与该请求关联的当前 session 会话，或者如果请求没有 session 会话，则创建一个。 |
| **HttpSession getSession(boolean create)** | 返回与该请求关联的当前 HttpSession，或者如果没有当前会话，且创建是真的，则返回一个新的 session 会话。 |
| **Locale getLocale()** | 基于 Accept-Language 头，返回客户端接受内容的首选的区域设置。 |
| **Object getAttribute(String name)** | 以对象形式返回已命名属性的值，如果没有给定名称的属性存在，则返回 null。 |
| **ServletInputStream getInputStream()** | 使用 ServletInputStream，以二进制数据形式检索请求的主体。 |
| **String getAuthType()** | 返回用于保护 Servlet 的身份验证方案的名称，例如，"BASIC" 或 "SSL"，如果JSP没有受到保护则返回 null。 |
| **String getCharacterEncoding()** | 返回请求主体中使用的字符编码的名称。 |
| **String getContentType()** | 返回请求主体的 MIME 类型，如果不知道类型则返回 null。 |
| **String getContextPath()** | 返回指示请求上下文的请求 URI 部分。 |
| **String getHeader(String name)** | 以字符串形式返回指定的请求头的值。 |
| **String getMethod()** | 返回请求的 HTTP 方法的名称，例如，GET、POST 或 PUT。 |
| | |

| | |
|---|---|
| **String getParameter(String name)** | 以字符串形式返回请求参数的值，或者如果参数不存在则返回 null。 |
| **String getPathInfo()** | 当请求发出时，返回与客户端发送的 URL 相关的任何额外的路径信息。 |
| **String getProtocol()** | 返回请求协议的名称和版本。 |
| **String getQueryString()** | 返回包含在路径后的请求 URL 中的查询字符串。 |
| **String getRemoteAddr()** | 返回发送请求的客户端的互联网协议（IP）地址。 |
| **String getRemoteHost()** | 返回发送请求的客户端的完全限定名称。 |
| **String getRemoteUser()** | 如果用户已通过身份验证，则返回发出请求的登录用户，或者如果用户未通过身份验证，则返回 null。 |
| **String getRequestURI()** | 从协议名称直到 HTTP 请求的第一行的查询字符串中，返回该请求的 URL 的一部分。 |
| **String getRequestedSessionId()** | 返回由客户端指定的 session 会话 ID。 |
| **String getServletPath()** | 返回调用 JSP 的请求的 URL 的一部分。 |
| **String[] getParameterValues(String name)** | 返回一个字符串对象的数组，包含所有给定的请求参数的值，如果参数不存在则返回 null。 |
| **boolean isSecure()** | 返回一个布尔值，指示请求是否使用安全通道，如 HTTPS。 |
| **int getContentLength()** | 以字节为单位返回请求主体的长度，并提供输入流，或者如果长度未知则返回 -1。 |
| **int getIntHeader(String name)** | 返回指定的请求头的值为一个 int 值。 |
| **int getServerPort()** | 返回接收到这个请求的端口号。 |

# HTTP Header 请求实例

下面的实例使用 HttpServletRequest 的 **getHeaderNames()** 方法读取 HTTP 头信息。该方法返回一个枚举，包含与当前的 HTTP 请求相关的头信息。

一旦我们有一个枚举，我们可以以标准方式循环枚举，使用 *hasMoreElements()* 方法来确定何时停止，使用 *nextElement()* 方法来获取每个参数的名称。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 扩展 HttpServlet 类
public class DisplayHeader extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "HTTP Header 请求实例";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n"+
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
        "<tr bgcolor=\"#949494\">\n" +
        "<th>Header 名称</th><th>Header 值</th>\n"+
        "</tr>\n");

      Enumeration headerNames = request.getHeaderNames();

      while(headerNames.hasMoreElements()) {
         String paramName = (String)headerNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n");
         String paramValue = request.getHeader(paramName);
         out.println("<td> " + paramValue + "</td></tr>\n");
      }
      out.println("</table>\n</body></html>");
  }
  // 处理 POST 方法请求的方法
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
  }
}
```

现在，调用上面的 Servlet 会产生以下结果：

```
<h1>HTTP Header 请求实例</h1>

<table>

<tbody>

<tr bgcolor="#949494"><th>Header 名称</th><th>Header 值</th></tr>

<tr><td>accept</td><td>*/*</td></tr>

<tr><td>accept-language</td><td>en-us</td></tr>

<tr><td>user-agent</td><td>Mozilla/4.0 (compatible; MSIE 7.0; Windo

<tr><td>accept-encoding</td><td>gzip, deflate</td></tr>

<tr><td>host</td><td>localhost:8080</td></tr>

<tr><td>connection</td><td>Keep-Alive</td></tr>

<tr><td>cache-control</td><td>no-cache</td></tr>

</tbody>

</table>
```

# Servlet 服务器 HTTP 响应

正如前面的章节中讨论的那样，当一个 Web 服务器响应一个 HTTP 请求时，响应通常包括一个状态行、一些响应报头、一个空行和文档。一个典型的响应如下所示：

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```

状态行包括 HTTP 版本（在本例中为 HTTP/1.1）、一个状态码（在本例中为 200）和一个对应于状态码的短消息（在本例中为 OK）。

下表总结了从 Web 服务器端返回到浏览器的最有用的 HTTP 1.1 响应报头，您会在 Web 编程中频繁地使用它们：

| 头信息 | 描述 |
|---|---|
| Allow | 这个头信息指定服务器支持的请求方法（GET、POST 等）。 |
| Cache-Control | 这个头信息指定响应文档在何种情况下可以安全地缓存。可能的值有：**public**、**private** 或 **no-cache** 等。Public 意味着文档是可缓存，Private 意味着文档是单个用户私用文档，且只能存储在私有（非共享）缓存中，no-cache 意味着文档不应被缓存。 |
| Connection | 这个头信息指示浏览器是否使用持久 HTTP 连接。值 **close** 指示浏览器不使用持久 HTTP 连接，值 **keep-alive** 意味着使用持久连接。 |
| Content-Disposition | 这个头信息可以让您请求浏览器要求用户以给定名称的文件把响应保存到磁盘。 |
| Content-Encoding | 在传输过程中，这个头信息指定页面的编码方式。 |
| Content-Language | 这个头信息表示文档编写所使用的语言。例如，en、en-us、ru 等。 |
| Content-Length | 这个头信息指示响应中的字节数。只有当浏览器使用持久（keep-alive）HTTP 连接时才需要这些信息。 |
| Content-Type | 这个头信息提供了响应文档的 MIME（Multipurpose Internet Mail Extension）类型。 |
| Expires | 这个头信息指定内容过期的时间，在这之后内容不再被缓存。 |
| Last-Modified | 这个头信息指示文档的最后修改时间。然后，客户端可以缓存文件，并在以后的请求中通过 **If-Modified-Since** 请求头信息提供一个日期。 |
| Location | 这个头信息应被包含在所有的带有状态码的响应中。在 300s 内，这会通知浏览器文档的地址。浏览器会自动重新连接到这个位置，并获取新的文档。 |
| Refresh | 这个头信息指定浏览器应该如何尽快请求更新的页面。您可以指定页面刷新的秒数。 |
| Retry-After | 这个头信息可以与 503（Service Unavailable 服务不可用）响应配合使用，这会告诉客户端多久就可以重复它的请求。 |
| Set-Cookie | 个头信息指定一个与页面关联的 cookie。 |

## 设置 HTTP 响应报头的方法

下面的方法可用于在 Servlet 程序中设置 HTTP 响应报头。这些方法通过 *HttpServletResponse* 对象可用。

| 方法 | 描述 |
| --- | --- |
| **String encodeRedirectURL(String url)** | 为 sendRedirect 方法中使用的指定的 URL 进行编码，或者如果编码不是必需的，则返回 URL 未改变。 |
| **String encodeURL(String url)** | 对包含 session 会话 ID 的指定 URL 进行编码，或者如果编码不是必需的，则返回 URL 未改变。 |
| **boolean containsHeader(String name)** | 返回一个布尔值，指示是否已经设置已命名的响应报头。 |
| **boolean isCommitted()** | 返回一个布尔值，指示响应是否已经提交。 |
| **void addCookie(Cookie cookie)** | 把指定的 cookie 添加到响应。 |
| **void addDateHeader(String name, long date)** | 添加一个带有给定的名称和日期值的响应报头。 |
| **void addHeader(String name, String value)** | 添加一个带有给定的名称和值的响应报头。 |
| **void addIntHeader(String name, int value)** | 添加一个带有给定的名称和整数值的响应报头。 |
| **void flushBuffer()** | 强制任何在缓冲区中的内容被写入到客户端。 |
| **void reset()** | 清除缓冲区中存在的任何数据，包括状态码和头。 |
| **void resetBuffer()** | 清除响应中基础缓冲区的内容，不清除状态码和头。 |
| **void sendError(int sc)** | 使用指定的状态码发送错误响应到客户端，并清除缓冲区。 |
| **void sendError(int sc, String msg)** | 使用指定的状态发送错误响应到客户端。 |
| **void sendRedirect(String location)** | 使用指定的重定向位置 URL 发送临时重定向响应到客户端。 |
| **void setBufferSize(int size)** | 为响应主体设置首选的缓冲区大小。 |
| **void setCharacterEncoding(String charset)** | 设置被发送到客户端的响应的字符编码（MIME 字符集）例如，UTF-8。 |
| **void setContentLength(int len)** | 设置在 HTTP Servlet 响应中的内容主体的长度，该方法设置 HTTP Content-Length |

| | |
|---|---|
| **len)** | 头。 |
| **void setContentType(String type)** | 如果响应还未被提交，设置被发送到客户端的响应的内容类型。 |
| **void setDateHeader(String name, long date)** | 设置一个带有给定的名称和日期值的响应报头。 |
| **void setHeader(String name, String value)** | 设置一个带有给定的名称和值的响应报头。 |
| **void setIntHeader(String name, int value)** | 设置一个带有给定的名称和整数值的响应报头。 |
| **void setLocale(Locale loc)** | 如果响应还未被提交，设置响应的区域。 |
| **void setStatus(int sc)** | 为该响应设置状态码。 |

## HTTP Header 响应实例

您已经在前面的实例中看到 setContentType() 方法，下面的实例也使用了同样的方法，此外，我们会用 **setIntHeader()** 方法来设置 **Refresh** 头。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 扩展 HttpServlet 类
public class Refresh extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置刷新自动加载时间为 5 秒
      response.setIntHeader("Refresh", 5);

      // 设置响应内容类型
      response.setContentType("text/html");

      // Get current time
      Calendar calendar = new GregorianCalendar();
      String am_pm;
      int hour = calendar.get(Calendar.HOUR);
      int minute = calendar.get(Calendar.MINUTE);
      int second = calendar.get(Calendar.SECOND);
      if(calendar.get(Calendar.AM_PM) == 0)
```

```
        else
          am_pm = "PM";

        String CT = hour+":"+ minute +":"+ second +" "+ am_pm;

        PrintWriter out = response.getWriter();
        String title = "自动刷新 Header 设置";
        String docType =
        "<!doctype html public \"-//w3c//dtd html 4.0 " +
        "transitional//en\">\n";
        out.println(docType +
          "<html>\n" +
          "<head><title>" + title + "</title></head>\n"+
          "<body bgcolor=\"#f0f0f0\">\n" +
          "<h1 align=\"center\">" + title + "</h1>\n" +
          "<p>当前时间是：" + CT + "</p>\n");
    }
    // 处理 POST 方法请求的方法
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
       doGet(request, response);
    }
}
```

现在，调用上面的 Servlet，每隔 5 秒会显示当前系统时间。只要运行 Servlet 并稍等片刻，即可看到如下的结果：

```
<h1>自动刷新 Header 设置</h1>

当前时间是：9:44:50 PM
```

# Servlet HTTP 状态码

HTTP 请求和 HTTP 响应消息的格式是类似的，结构如下：

- 初始状态行 + 回车换行符（回车+换行）
- 零个或多个标题行+回车换行符
- 一个空白行，即回车换行符
- 一个可选的消息主体，比如文件、查询数据或查询输出

例如，服务器的响应头如下所示：

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```

状态行包括 HTTP 版本（在本例中为 HTTP/1.1）、一个状态码（在本例中为 200）和一个对应于状态码的短消息（在本例中为 OK）。

以下是可能从 Web 服务器返回的 HTTP 状态码和相关的信息列表：

| 代码 | 消息 | 描述 |
|---|---|---|
| 100 | Continue | 只有请求的一部分已经被服务器接收，但只要它没有被拒绝，客户端应继续该请求。 |
| 101 | Switching Protocols | 服务器切换协议。 |
| 200 | OK | 请求成功。 |
| 201 | Created | 该请求是完整的，并创建一个新的资源。 |
| 202 | Accepted | 该请求被接受处理，但是该处理是不完整的。 |
| 203 | Non-authoritative Information | |

| | | |
|---|---|---|
| 204 | No Content | |
| 205 | Reset Content | |
| 206 | Partial Content | |
| 300 | Multiple Choices | 链接列表。用户可以选择一个链接，进入到该位置。最多五个地址。 |
| 301 | Moved Permanently | 所请求的页面已经转移到一个新的 URL。 |
| 302 | Found | 所请求的页面已经临时转移到一个新的 URL。 |
| 303 | See Other | 所请求的页面可以在另一个不同的 URL 下被找到。 |
| 304 | Not Modified | |
| 305 | Use Proxy | |
| 306 | *Unused* | 在以前的版本中使用该代码。现在已不再使用它，但代码仍被保留。 |
| 307 | Temporary Redirect | 所请求的页面已经临时转移到一个新的 URL。 |
| 400 | Bad Request | 服务器不理解请求。 |
| 401 | Unauthorized | 所请求的页面需要用户名和密码。 |
| 402 | Payment Required | 您还不能使用该代码。 |
| 403 | Forbidden | 禁止访问所请求的页面。 |
| 404 | Not Found | 服务器无法找到所请求的页面。. |
| 405 | Method Not Allowed | 在请求中指定的方法是不允许的。 |
| 406 | Not Acceptable | 服务器只生成一个不被客户端接受的响应。 |
| 407 | Proxy Authentication Required | 在请求送达之前，您必须使用代理服务器的验证。 |
| 408 | Request Timeout | 请求需要的时间比服务器能够等待的时间长，超时。 |
| 409 | Conflict | 请求因为冲突无法完成。 |
| 410 | Gone | 所请求的页面不再可用。 |
| | | |

| | | |
|---|---|---|
| 411 | Length Required | "Content-Length" 未定义。服务器无法处理客户端发送的不带 Content-Length 的请求信息。 |
| 412 | Precondition Failed | 请求中给出的先决条件被服务器评估为 false。 |
| 413 | Request Entity Too Large | 服务器不接受该请求，因为请求实体过大。 |
| 414 | Request-url Too Long | 服务器不接受该请求，因为 URL 太长。当您转换一个 "post" 请求为一个带有长的查询信息的 "get" 请求时发生。 |
| 415 | Unsupported Media Type | 服务器不接受该请求，因为媒体类型不被支持。 |
| 417 | Expectation Failed | |
| 500 | Internal Server Error | 未完成的请求。服务器遇到了一个意外的情况。 |
| 501 | Not Implemented | 未完成的请求。服务器不支持所需的功能。 |
| 502 | Bad Gateway | 未完成的请求。服务器从上游服务器收到无效响应。 |
| 503 | Service Unavailable | 未完成的请求。服务器暂时超载或死机。 |
| 504 | Gateway Timeout | 网关超时。 |
| 505 | HTTP Version Not Supported | 服务器不支持"HTTP协议"版本。 |

# 设置 HTTP 状态代码的方法

下面的方法可用于在 Servlet 程序中设置 HTTP 状态码。这些方法通过 *HttpServletResponse* 对象可用。

| 方法 | 描述 |
|---|---|
| **public void setStatus ( int statusCode )** | 该方法设置一个任意的状态码。setStatus 方法接受一个 int（状态码）作为参数。如果您的反应包含了一个特殊的状态码和文档，请确保在使用 *PrintWriter* 实际返回任何内容之前调用 setStatus。 |
| **public void sendRedirect(String url)** | 该方法生成一个 302 响应，连同一个带有新文档 URL 的 *Location* 头。 |
| **public void sendError(int code, String message)** | 该方法发送一个状态码（通常为 404），连同一个在 HTML 文档内部自动格式化并发送到客户端的短消息。 |

# HTTP 状态码实例

下面的例子把 407 错误代码发送到客户端浏览器，浏览器会显示 "Need authentication!!!" 消息。

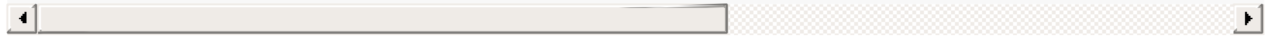
```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 扩展 HttpServlet 类
public class showError extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置错误代码和原因
      response.sendError(407, "Need authentication!!!" );
  }
  // 处理 POST 方法请求的方法
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
  }
}
```

现在，调用上面的 Servlet 将显示以下结果：

```
<h2>HTTP Status 407 - Need authentication!!!</h2>
<b>type</b> Status report
<b>message</b> <u>Need authentication!!!</u>
<b>description</b> <u>The client must first authenticate itself wit
<h3>Apache Tomcat/5.5.29</h3>
```



# Servlet 编写过滤器

Servlet 过滤器是可用于 Servlet 编程的 Java 类，有以下目的：

- 在客户端的请求访问后端资源之前，拦截这些请求。
- 在服务器的响应发送回客户端之前，处理这些响应。

根据规范建议的各种类型的过滤器：

- 身份验证过滤器（Authentication Filters）。
- 数据压缩过滤器（Data compression Filters）。
- 加密过滤器（Encryption Filters）。
- 触发资源访问事件过滤器。
- 图像转换过滤器（Image Conversion Filters）。
- 日志记录和审核过滤器（Logging and Auditing Filters）。
- MIME-TYPE 链过滤器（MIME-TYPE Chain Filters）。
- 标记化过滤器（Tokenizing Filters）。
- XSL/T 过滤器（XSL/T Filters），转换 XML 内容。

过滤器被部署在部署描述符文件 **web.xml** 中，然后映射到您的应用程序的部署描述符中的 Servlet 名称或 URL 模式。

当 Web 容器启动 Web 应用程序时，它会为您在部署描述符中声明的每一个过滤器创建一个实例。该过滤器执行的顺序是按它们在部署描述符中声明的顺序。

## Servlet 过滤器方法

过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类。javax.servlet.Filter 接口定义了三个方法：

| 方法 | 描述 |
|---|---|
| **public void doFilter (ServletRequest, ServletResponse, FilterChain)** | 该方法在每次一个请求/响应对因客户端在链的末端请求资源而通过链传递时由容器调用。 |
| **public void init(FilterConfig filterConfig)** | 该方法由 Web 容器调用，指示一个过滤器被放入服务。 |
| **public void destroy()** | 该方法由 Web 容器调用，指示一个过滤器被取出服务。 |

## Servlet 过滤器实例

以下是 Servlet 过滤器的实例，将输出客户端的 IP 地址和当前的日期时间。本实例让您对 Servlet 过滤器有基本的了解，您可以使用相同的概念编写更复杂的过滤器应用程序：

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 实现 Filter 类
public class LogFilter implements Filter  {
   public void  init(FilterConfig config) 
                         throws ServletException{
      // 获取初始化参数
      String testParam = config.getInitParameter("test-param"); 

      // 输出初始化参数
      System.out.println("Test Param: " + testParam); 
   }
   public void  doFilter(ServletRequest request, 
                 ServletResponse response,
                 FilterChain chain) 
                 throws java.io.IOException, ServletException {

      // 获取客户机的 IP 地址   
      String ipAddress = request.getRemoteAddr();

      // 记录 IP 地址和当前时间戳
      System.out.println("IP "+ ipAddress + ", Time "
                                       + new Date().toString());

      // 把请求传回过滤链
      chain.doFilter(request,response);
   }
   public void destroy( ){
      /* 在 Filter 实例被 Web 容器从服务移除之前调用 */
   }
}
```

以通常的方式编译 **LogFilter.java**，把您的类文件放入 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中。

# Web.xml 中的 Servlet 过滤器映射（Servlet Filter Mapping）

定义过滤器，然后映射到一个 URL 或 Servlet，这与定义 Servlet，然后映射到一个 URL 模式方式大致相同。在部署描述符文件 **web.xml** 中为 filter 标签创建下面的条目：

```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
   <init-param>
      <param-name>test-param</param-name>
      <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

上述过滤器适用于所有的 Servlet，因为我们在配置中指定 **/*** 。如果您只想在少数的 Servlet 上应用过滤器，您可以指定一个特定的 Servlet 路径。

现在试着以常用的方式调用任何 Servlet，您将会看到在 Web 服务器中生成的日志。您也可以使用 Log4J 记录器来把上面的日志记录到一个单独的文件中。

## 使用多个过滤器

Web 应用程序可以根据特定的目的定义若干个不同的过滤器。假设您定义了两个过滤器 *AuthenFilter* 和 *LogFilter*。您需要创建一个如下所述的不同的映射，其余的处理与上述所讲解的大致相同：

```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
   <init-param>
      <param-name>test-param</param-name>
      <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>

<filter>
   <filter-name>AuthenFilter</filter-name>
   <filter-class>AuthenFilter</filter-class>
   <init-param>
      <param-name>test-param</param-name>
      <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>

<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>AuthenFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

# 过滤器的应用顺序

web.xml 中的 filter-mapping 元素的顺序决定了 Web 容器应用过滤器到 Servlet 的顺序。若要反转过滤器的顺序，您只需要在 web.xml 文件中反转 filter-mapping 元素即可。

例如，上面的实例将先应用 LogFilter，然后再应用 AuthenFilter，但是下面的实例将颠倒这个顺序：

```
<filter-mapping>
   <filter-name>AuthenFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

# Servlet 异常处理

当一个 Servlet 抛出一个异常时，Web 容器在使用了 exception-type 元素的 **web.xml** 中搜索与抛出异常类型相匹配的配置。

您必须在 web.xml 中使用 **error-page** 元素来指定对特定异常 或 HTTP 状态码 作出相应的 Servlet 调用。

## web.xml 配置

假设，有一个 *ErrorHandler* 的 Servelt 在任何已定义的异常或错误出现时被调用。以下将是在 web.xml 中创建的项。

```
<!-- servlet 定义 -->
<servlet>
        <servlet-name>ErrorHandler</servlet-name>
        <servlet-class>ErrorHandler</servlet-class>
</servlet>
<!-- servlet 映射 -->
<servlet-mapping>
        <servlet-name>ErrorHandler</servlet-name>
        <url-pattern>/ErrorHandler</url-pattern>
</servlet-mapping>

<!-- error-code 相关的错误页面 -->
<error-page>
    <error-code>404</error-code>
    <location>/ErrorHandler</location>
</error-page>
<error-page>
    <error-code>403</error-code>
    <location>/ErrorHandler</location>
</error-page>

<!-- exception-type 相关的错误页面 -->
<error-page>
    <exception-type>
          javax.servlet.ServletException
    </exception-type >
    <location>/ErrorHandler</location>
</error-page>

<error-page>
    <exception-type>java.io.IOException</exception-type >
    <location>/ErrorHandler</location>
</error-page>
```

如果您想对所有的异常有一个通用的错误处理程序，那么应该定义下面的 error-page，而不是为每个异常定义单独的 error-page 元素：

```
<error-page>
    <exception-type>java.lang.Throwable</exception-type >
    <location>/ErrorHandler</location>
</error-page>
```

以下是关于上面的 web.xml 异常处理要注意的点：

- Servelt ErrorHandler 与其他的 Servelt 的定义方式一样，且在 web.xml 中进行配置。
- 如果有错误状态代码出现，不管为 404（Not Found 未找到）或 403（Forbidden 禁止），则会调用 ErrorHandler 的 Servlet。
- 如果 Web 应用程序抛出 *ServletException* 或 *IOException*，那么 Web 容器会调用 ErrorHandler 的 Servlet。
- 您可以定义不同的错误处理程序来处理不同类型的错误或异常。上面的实例是非常通用的，希望您能通过实例理解基本的概念。

## 请求属性 - 错误/异常

以下是错误处理的 Servlet 可以访问的请求属性列表，用来分析错误/异常的性质。

| 属性 | 描述 |
|---|---|
| **javax.servlet.error.status_code** | 该属性给出状态码，状态码可被存储，并在存储为 java.lang.Integer 数据类型后可被分析。 |
| **javax.servlet.error.exception_type** | 该属性给出异常类型的信息，异常类型可被存储，并在存储为 java.lang.Class 数据类型后可被分析。 |
| **javax.servlet.error.message** | 该属性给出确切错误消息的信息，信息可被存储，并在存储为 java.lang.String 数据类型后可被分析。 |
| **javax.servlet.error.request_uri** | 该属性给出有关 URL 调用 Servlet 的信息，信息可被存储，并在存储为 java.lang.String 数据类型后可被分析。 |
| **javax.servlet.error.exception** | 该属性给出异常产生的信息，信息可被存储，并在存储为 java.lang.Throwable 数据类型后可被分析。 |
| **javax.servlet.error.servlet_name** | 该属性给出 Servlet 的名称，名称可被存储，并在存储为 java.lang.String 数据类型后可被分析。 |

# Servlet 错误处理程序实例

以下是 Servlet 实例，将应对任何您所定义的错误或异常发生时的错误处理程序。

本实例让您对 Servlet 中的异常处理有基本的了解，您可以使用相同的概念编写更复杂的异常处理应用程序：

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 扩展 HttpServlet 类
public class ErrorHandler extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 分析 Servlet 异常
      Throwable throwable = (Throwable)
      request.getAttribute("javax.servlet.error.exception");
      Integer statusCode = (Integer)
      request.getAttribute("javax.servlet.error.status_code");
      String servletName = (String)
      request.getAttribute("javax.servlet.error.servlet_name");
      if (servletName == null){
         servletName = "Unknown";
      }
      String requestUri = (String)
      request.getAttribute("javax.servlet.error.request_uri");
      if (requestUri == null){
         requestUri = "Unknown";
      }

      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Error/Exception Information";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n");

      if (throwable == null && statusCode == null){
```

```
          out.println("<h2>Error information is missing</h2>");
          out.println("Please return to the <a href=\"" +
            response.encodeURL("http://localhost:8080/") +
            "\">Home Page</a>.");
       }else if (statusCode != null){
          out.println("The status code : " + statusCode);
       }else{
          out.println("<h2 class="tutheader">Error information</h2>'
          out.println("Servlet Name : " + servletName +
                              "</br></br>");
          out.println("Exception Type : " +
                              throwable.getClass( ).getName( ) +
                              "</br></br>");
          out.println("The request URI: " + requestUri +
                              "<br><br>");
          out.println("The exception message: " +
                                  throwable.getMessage( ));
       }
       out.println("</body>");
       out.println("</html>");
   }
   // 处理 POST 方法请求的方法
   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
       throws ServletException, IOException {
      doGet(request, response);
   }
}
```

以通常的方式编译 **ErrorHandler.java**，把您的类文件放入<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中。

让我们在 web.xml 文件中添加如下配置来处理异常：

```
<servlet>
        <servlet-name>ErrorHandler</servlet-name>
        <servlet-class>ErrorHandler</servlet-class>
</servlet>
<!-- servlet mappings -->
<servlet-mapping>
        <servlet-name>ErrorHandler</servlet-name>
        <url-pattern>/ErrorHandler</url-pattern>
</servlet-mapping>
<error-page>
    <error-code>404</error-code>
    <location>/ErrorHandler</location>
</error-page>
<error-page>
    <exception-type>java.lang.Throwable</exception-type >
    <location>/ErrorHandler</location>
</error-page>
```

现在，尝试使用一个会产生异常的 Servlet，或者输入一个错误的 URL，这将触发 Web 容器调用 **ErrorHandler** 的 Servlet，并显示适当的消息。例如，如果您输入了一个错误的 URL，那么它将显示下面的结果：

```
The status code : 404
```

上面的代码在某些 Web 浏览器中可能无法正常工作。因此，请尽量尝试使用 Mozilla 和 Safari 浏览器，在这两种浏览器中它们应该能够正常工作。

# Servlet Cookies 处理

Cookies 是存储在客户端计算机上的文本文件，并保留了各种跟踪信息。Java Servlet 显然支持 HTTP Cookies。

识别返回用户包括三个步骤：

- 服务器脚本向浏览器发送一组 Cookies。例如：姓名、年龄或识别号码等。
- 浏览器将这些信息存储在本地计算机上，以备将来使用。
- 当下一次浏览器向 Web 服务器发送任何请求时，浏览器会把这些 Cookies 信息发送到服务器，服务器将使用这些信息来识别用户。

本章将向您讲解如何设置或重置 Cookies，如何访问它们，以及如何将它们删除。

## Cookie 剖析

Cookies 通常设置在 HTTP 头信息中（虽然 JavaScript 也可以直接在浏览器上设置一个 Cookie）。设置 Cookie 的 Servlet 会发送如下的头信息：

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
                 path=/; domain=w3cschool.cc
Connection: close
Content-Type: text/html
```

正如您所看到的，Set-Cookie 头包含了一个名称值对、一个 GMT 日期、一个路径和一个域。名称和值会被 URL 编码。expires 字段是一个指令，告诉浏览器在给定的时间和日期之后"忘记"该 Cookie。

如果浏览器被配置为存储 Cookies，它将会保留此信息直到到期日期。如果用户的浏览器指向任何匹配该 Cookie 的路径和域的页面，它会重新发送 Cookie 到服务器。浏览器的头信息可能如下所示：

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

Servlet 就能够通过请求方法 *request.getCookies()* 访问 Cookie，该方法将返回一个 *Cookie* 对象的数组。

## Servlet Cookies 方法

以下是在 Servlet 中操作 Cookies 时可使用的有用的方法列表。

| 方法 | 描述 |
|---|---|
| **public void setDomain(String pattern)** | 该方法设置 cookie 适用的域，例如 w3cschool.cc。 |
| **public String getDomain()** | 该方法获取 cookie 适用的域，例如 w3cschool.cc。 |
| **public void setMaxAge(int expiry)** | 该方法设置 cookie 过期的时间（以秒为单位）。如果不这样设置，cookie 只会在当前 session 会话中持续有效。 |
| **public int getMaxAge()** | 该方法返回 cookie 的最大生存周期（以秒为单位），默认情况下，-1 表示 cookie 将持续下去，直到浏览器关闭。 |
| **public String getName()** | 该方法返回 cookie 的名称。名称在创建后不能改变。 |
| **public void setValue(String newValue)** | 该方法设置与 cookie 关联的值。 |
| **public String getValue()** | 该方法获取与 cookie 关联的值。 |
| **public void setPath(String uri)** | 该方法设置 cookie 适用的路径。如果您不指定路径，与当前页面相同目录下的（包括子目录下的）所有 URL 都会返回 cookie。 |
| **public String getPath()** | 该方法获取 cookie 适用的路径。 |
| **public void setSecure(boolean flag)** | 该方法设置布尔值，表示 cookie 是否应该只在加密的（即 SSL）连接上发送。 |
| **public void setComment(String purpose)** | 该方法规定了描述 cookie 目的的注释。该注释在浏览器向用户呈现 cookie 时非常有用。 |
| **public String getComment()** | 该方法返回了描述 cookie 目的的注释，如果 cookie 没有注释则返回 null。 |

# 通过 Servlet 设置 Cookies

通过 Servlet 设置 Cookies 包括三个步骤：

**(1)** 创建一个 **Cookie** 对象：您可以调用带有 cookie 名称和 cookie 值的 Cookie 构造函数，cookie 名称和 cookie 值都是字符串。

```
Cookie cookie = new Cookie("key","value");
```

请记住，无论是名字还是值，都不应该包含空格或以下任何字符：

```
[ ] ( ) = , " / ? @ : ;
```

**(2)** 设置最大生存周期：您可以使用 setMaxAge 方法来指定 cookie 能够保持有效的时间（以秒为单位）。下面将设置一个最长有效期为 24 小时的 cookie。

```
cookie.setMaxAge(60*60*24);
```

**(3)** 发送 **Cookie** 到 **HTTP** 响应头：您可以使用 **response.addCookie** 来添加 HTTP 响应头中的 Cookies，如下所示：

```
response.addCookie(cookie);
```

## 实例

让我们修改我们的 表单数据实例，为名字和姓氏设置 Cookies。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 为名字和姓氏创建 Cookies
      Cookie firstName = new Cookie("first_name",
                      request.getParameter("first_name"));
      Cookie lastName = new Cookie("last_name",
                      request.getParameter("last_name"));

      // 为两个 Cookies 设置过期日期为 24 小时后
      firstName.setMaxAge(60*60*24);
      lastName.setMaxAge(60*60*24);

      // 在响应头中添加两个 Cookies
      response.addCookie( firstName );
      response.addCookie( lastName );

      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "设置 Cookies 实例";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<ul>\n" +
                "  <li><b>名字</b>："
                + request.getParameter("first_name") + "\n" +
                "  <li><b>姓氏</b>："
                + request.getParameter("last_name") + "\n" +
                "</ul>\n" +
                "</body></html>");
  }
}
```

编译上面的 Servlet **HelloForm**，并在 web.xml 文件中创建适当的条目，最后尝试下面的 HTML 页面来调用 Servlet。

```
<html>
<body>
<form action="HelloForm" method="GET">
名字：<input type="text" name="first_name">
<br />
姓氏：<input type="text" name="last_name" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

保存上面的 HTML 内容到文件 hello.htm 中，并把它放在 <Tomcat-installation-directory>/webapps/ROOT 目录中。当您访问 *http://localhost:8080/Hello.htm* 时，上面表单的实际输出如下所示：

名字： [ ] 姓氏： [ ] [提交]

尝试输入名字和姓氏，然后点击"提交"按钮，名字和姓氏将显示在屏幕上，同时会设置 firstName 和 lastName 这两个 Cookies，当下次您按下提交按钮时，会被这两个 Cookies 传回到服务器。

下一节会讲解如何在 Web 应用程序中访问这些 Cookies。

# 通过 Servlet 读取 Cookies

要读取 Cookies，您需要通过调用 *HttpServletRequest* 的 **getCookies( )** 方法创建一个 *javax.servlet.http.Cookie* 对象的数组。然后循环遍历数组，并使用 getName() 和 getValue() 方法来访问每个 cookie 和关联的值。

# 实例

让我们读取上面的实例中设置的 Cookies

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class ReadCookies extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      Cookie cookie = null;
      Cookie[] cookies = null;
      // 获取与该域相关的 Cookies 的数组
      cookies = request.getCookies();

      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading Cookies Example";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" );
      if( cookies != null ){
         out.println("<h2>查找 Cookies 名称和值</h2>");
         for (int i = 0; i < cookies.length; i++){
            cookie = cookies[i];
            out.print("名称：" + cookie.getName( ) + ", ");
            out.print("值：" + cookie.getValue( )+" <br/>");
         }
      }else{
          out.println(
            "<h2 class="tutheader">未找到 Cookies</h2>");
      }
      out.println("</body>");
      out.println("</html>");
   }
}
```

编译上面的 Servlet **ReadCookies**，并在 web.xml 文件中创建适当的条目。如果您已经设置了 first*name cookie* 为 *"John"*，*last_name cookie* 为 *"Player"*，尝试运行 _*http://localhost:8080/ReadCookies*，将显示如下结果：

```
<tr><td>
<h2>查找 Cookies 名称和值</h2>
名称：first_name，值：John
名称：last_name，值：Player</td></tr>
```

## 通过 Servlet 删除 Cookies

删除 Cookies 是非常简单的。如果您想删除一个 cookie，那么您只需要按照以下三个步骤进行：

- 读取一个现有的 cookie，并把它存储在 Cookie 对象中。
- 使用 **setMaxAge()** 方法设置 cookie 的年龄为零，来删除现有的 cookie。
- 把这个 cookie 添加到响应头。

## 实例

下面的例子将删除现有的名为 "first_name" 的 cookie，当您下次运行 ReadCookies 的 Servlet 时，它会返回 first_name 为空值。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class DeleteCookies extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      Cookie cookie = null;
      Cookie[] cookies = null;
      // 获取与该域相关的 Cookies 的数组
      cookies = request.getCookies();

      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Delete Cookies Example";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" );
       if( cookies != null ){
         out.println("<h2>Cookies 名称和值</h2>");
         for (int i = 0; i < cookies.length; i++){
            cookie = cookies[i];
            if((cookie.getName( )).compareTo("first_name") == 0 ){
                 cookie.setMaxAge(0);
                 response.addCookie(cookie);
                 out.print("已删除的 cookie：" +
                              cookie.getName( ) + "<br/>");
            }
            out.print("名称：" + cookie.getName( ) + ", ");
            out.print("值：" + cookie.getValue( )+" <br/>");
         }
      }else{
          out.println(
            "<h2 class="tutheader">No cookies founds</h2>");
      }
      out.println("</body>");
      out.println("</html>");
   }
}
```

编译上面的 Servlet **DeleteCookies**，并在 web.xml 文件中创建适当的条目。现在运行 *http://localhost:8080/DeleteCookies*，将显示如下结果：

```
<h2>Cookies 名称和值</h2>
已删除的 cookie：first_name
名称：first_name，值：John
名称：last_name，值：Player
```

现在尝试运行 *http://localhost:8080/ReadCookies*，它将只显示一个 cookie，如下所示：

```
<h2>查找 Cookies 名称和值</h2>
名称：last_name，值：Player
```

您可以手动在 Internet Explorer 中删除 Cookies。在"工具"菜单，选择"Internet 选项"。如果要删除所有的 Cookies，请按"删除 Cookies"。

# Servlet Session 跟踪

HTTP 是一种"无状态"协议，这意味着每次客户端检索网页时，客户端打开一个单独的连接到 Web 服务器，服务器会自动不保留之前客户端请求的任何记录。

但是仍然有以下三种方式来维持 Web 客户端和 Web 服务器之间的 session 会话：

## Cookies

一个 Web 服务器可以分配一个唯一的 session 会话 ID 作为每个 Web 客户端的 cookie，对于客户端的后续请求可以使用接收到的 cookie 来识别。

这可能不是一个有效的方法，因为很多浏览器不支持 cookie，所以我们建议不要使用这种方式来维持 session 会话。

## 隐藏的表单字段

一个 Web 服务器可以发送一个隐藏的 HTML 表单字段，以及一个唯一的 session 会话 ID，如下所示：

```
<input type="hidden" name="sessionid" value="12345">
```

该条目意味着，当表单被提交时，指定的名称和值会被自动包含在 GET 或 POST 数据中。每次当 Web 浏览器发送回请求时，session_id 值可以用于保持不同的 Web 浏览器的跟踪。

这可能是一种保持 session 会话跟踪的有效方式，但是点击常规的超文本链接（<A HREF...>）不会导致表单提交，因此隐藏的表单字段也不支持常规的 session 会话跟踪。

## URL 重写

您可以在每个 URL 末尾追加一些额外的数据来标识 session 会话，服务器会把该 session 会话标识符与已存储的有关 session 会话的数据相关联。

例如，http://w3cschool.cc/file.htm;sessionid=12345，session 会话标识符被附加为 sessionid=12345，标识符可被 Web 服务器访问以识别客户端。

URL 重写是一种更好的维持 session 会话的方式，它在浏览器不支持 cookie 时能够很好地工作，但是它的缺点是会动态生成每个 URL 来为页面分配一个 session 会话 ID，即使是在很简单的静态 HTML 页面中也会如此。

# HttpSession 对象

除了上述的三种方式，Servlet 还提供了 HttpSession 接口，该接口提供了一种跨多个页面请求或访问网站时识别用户以及存储有关用户信息的方式。

Servlet 容器使用这个接口来创建一个 HTTP 客户端和 HTTP 服务器之间的 session 会话。会话持续一个指定的时间段，跨多个连接或页面请求。

您会通过调用 HttpServletRequest 的公共方法 **getSession()** 来获取 HttpSession 对象，如下所示：

```
HttpSession session = request.getSession();
```

你需要在向客户端发送任何文档内容之前调用 *request.getSession()*。下面总结了 HttpSession 对象中可用的几个重要的方法：

| 方法 | 描述 |
|---|---|
| **public Object getAttribute(String name)** | 该方法返回在该 session 会话中具有指定名称的对象，如果没有指定名称的对象，则返回 null。 |
| **public Enumeration getAttributeNames()** | 该方法返回 String 对象的枚举，String 对象包含所有绑定到该 session 会话的对象的名称。 |
| **public long getCreationTime()** | 该方法返回该 session 会话被创建的时间，自格林尼治标准时间 1970 年 1 月 1 日午夜算起，以毫秒为单位。 |
| **public String getId()** | 该方法返回一个包含分配给该 session 会话的唯一标识符的字符串。 |
| **public long getLastAccessedTime()** | 该方法返回客户端最后一次发送与该 session 会话相关的请求的时间自格林尼治标准时间 1970 年 1 月 1 日午夜算起，以毫秒为单位。 |
| **public int getMaxInactiveInterval()** | 该方法返回 Servlet 容器在客户端访问时保持 session 会话打开的最大时间间隔，以秒为单位。 |
| **public void invalidate()** | 该方法指示该 session 会话无效，并解除绑定到它上面的任何对象。 |
| **public boolean isNew(** | 如果客户端还不知道该 session 会话，或者如果客户选择不参入该 session 会话，则该方法返回 true。 |
| **public void removeAttribute(String name)** | 该方法将从该 session 会话移除指定名称的对象。 |
| **public void setAttribute(String name, Object value)** | 该方法使用指定的名称绑定一个对象到该 session 会话。 |
| **public void setMaxInactiveInterval(int interval)** | 该方法在 Servlet 容器指示该 session 会话无效之前，指定客户端请求之间的时间，以秒为单位。 |

# Session 跟踪实例

本实例说明了如何使用 HttpSession 对象获取 session 会话创建时间和最后访问时间。如果不存在 session 会话，我们将通过请求创建一个新的 session 会话。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;
import java.util.*;

// 扩展 HttpServlet 类
public class SessionTrack extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 如果不存在 session 会话，则创建一个 session 对象
      HttpSession session = request.getSession(true);
      // 获取 session 创建时间
      Date createTime = new Date(session.getCreationTime());
      // 获取该网页的最后一次访问时间
      Date lastAccessTime =
                        new Date(session.getLastAccessedTime());

      String title = "欢迎回到我的网站";
      Integer visitCount = new Integer(0);
      String visitCountKey = new String("visitCount");
      String userIDKey = new String("userID");
      String userID = new String("ABCD");

      // 检查网页上是否有新的访问者
      if (session.isNew()){
         title = "欢迎来到我的网站";
         session.setAttribute(userIDKey, userID);
      } else {
         visitCount = (Integer)session.getAttribute(visitCountKey);
         visitCount = visitCount + 1;
         userID = (String)session.getAttribute(userIDKey);
      }
      session.setAttribute(visitCountKey,  visitCount);

      // 设置响应内容类型
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                 "<h2 align=\"center\">Session 信息</h2>\n" +
                "<table border=\"1\" align=\"center\">\n" +
                "<tr bgcolor=\"#949494\">\n" +
                "  <th>Session 信息</th><th>值</th></tr>\n" +
                "<tr>\n" +
                "  <td>id</td>\n" +
```

```
                  "  <td>" + session.getId() + "</td></tr>\n" +
                  "<tr>\n" +
                  "  <td>Creation Time</td>\n" +
                  "  <td>" + createTime +
                  "  </td></tr>\n" +
                  "<tr>\n" +
                  "  <td>Time of Last Access</td>\n" +
                  "  <td>" + lastAccessTime +
                  "  </td></tr>\n" +
                  "<tr>\n" +
                  "  <td>User ID</td>\n" +
                  "  <td>" + userID +
                  "  </td></tr>\n" +
                  "<tr>\n" +
                  "  <td>Number of visits</td>\n" +
                  "  <td>" + visitCount + "</td></tr>\n" +
                  "</table>\n" +
                  "</body></html>");
  }
}
```

编译上面的 Servlet **SessionTrack**，并在 web.xml 文件中创建适当的条目。在浏览器地址栏输入 *http://localhost:8080/SessionTrack*，当您第一次运行时将显示如下结果：

```
<h1>欢迎来到我的网站</h1>

<h2>Session 信息</h2>

<table>

<tbody>

<tr bgcolor="#949494"><th>Session 信息</th><th>值</th></tr>

<tr><td>id</td><td>0AE3EC93FF44E3C525B4351B77ABB2D5</td></tr>

<tr><td>Creation Time</td><td>Tue Jun 08 17:26:40 GMT+04:00 2014</t

<tr><td>Time of Last Access</td><td>Tue Jun 08 17:26:40 GMT+04:00 2

<tr><td>User ID</td><td>ABCD</td></tr>

<tr><td>Number of visits</td><td>0</td></tr>

</tbody>

</table>
```

再次尝试运行相同的 Servlet，它将显示如下结果：

```
<h1>欢迎回到我的网站</h1>

<h2>Session 信息</h2>

<table>

<tbody>

<tr bgcolor="#949494"><th>Session 信息</th><th>值</th></tr>

<tr><td>id</td><td>0AE3EC93FF44E3C525B4351B77ABB2D5</td></tr>

<tr><td>Creation Time</td><td>Tue Jun 08 17:26:40 GMT+04:00 2014</t

<tr><td>Time of Last Access</td><td>Tue Jun 08 17:26:40 GMT+04:00 2

<tr><td>User ID</td><td>ABCD</td></tr>

<tr><td>Number of visits</td><td>1</td></tr>

</tbody>

</table>
```

# 删除 Session 会话数据

当您完成了一个用户的 session 会话数据，您有以下几种选择：

- 移除一个特定的属性：您可以调用 *public void removeAttribute(String name)* 方法来删除与特定的键相关联的值。 to delete the value associated with a particular key.
- 删除整个 **session** 会话：您可以调用 *public void invalidate()* 方法来丢弃整个 session 会话。
- 设置 **session** 会话过期时间：您可以调用 *public void setMaxInactiveInterval(int interval)* 方法来单独设置 session 会话超时。
- 注销用户：如果使用的是支持 servlet 2.4 的服务器，您可以调用 **logout** 来注销 Web 服务器的客户端，并把属于所有用户的所有 session 会话设置为无效。
- **web.xml** 配置：如果您使用的是 Tomcat，除了上述方法，您还可以在 web.xml 文件中配置 session 会话超时，如下所示：

```
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

上面实例中的超时时间是以分钟为单位，将覆盖 Tomcat 中默认的 30 分钟超时时间。

在一个 Servlet 中的 getMaxInactiveInterval() 方法会返回 session 会话的超时时间，以秒为单位。所以，如果在 web.xml 中配置 session 会话超时时间为 15 分钟，那么 getMaxInactiveInterval() 会返回 900。

# Servlet 数据库访问

本教程假定您已经了解了 JDBC 应用程序的工作方式。在您开始学习 Servlet 数据库访问之前，请确保您已经有适当的 JDBC 环境设置和数据库。

从基本概念下手，让我们来创建一个简单的表，并在表中创建几条记录。

## 创建数据库表

在测试数据库 **TEST** 中创建 **Employees** 表，请按以下步骤进行：

## 步骤 1：

打开命令行提示符（**Command Prompt**），并更改进入到安装目录，如下所示：

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

## 步骤 2：

登录到数据库，如下所示：

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ********
mysql>
```

## 步骤 3：

在测试数据库 **TEST** 中创建 **Employee** 表，如下所示：

```
mysql> use TEST;
mysql> create table Employees
    (
     id int not null,
     age int not null,
     first varchar (255),
     last varchar (255)
    );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

# 创建数据记录

最后，在 Employee 表中创建几条记录，如下所示：

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)

mysql>
```

# 访问数据库

下面的实例演示了如何使用 Servlet 访问 TEST 数据库。

```
// 加载必需的库
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class DatabaseAccess extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
```

```
   {
      // JDBC 驱动器名称和数据库的 URL
      static final String JDBC_DRIVER="com.mysql.jdbc.Driver";
      static final String DB_URL="jdbc:mysql://localhost/TEST";

      //  数据库的凭据
      static final String USER = "root";
      static final String PASS = "password";

      // 设置响应内容类型
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "数据库结果";
      String docType =
        "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";
         out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor=\"#f0f0f0\">\n" +
         "<h1 align=\"center\">" + title + "</h1>\n");
      try{
         // 注册 JDBC 驱动器
         Class.forName("com.mysql.jdbc.Driver");

         // 打开一个连接
         conn = DriverManager.getConnection(DB_URL,USER,PASS);

         // 执行 SQL 查询
         stmt = conn.createStatement();
         String sql;
         sql = "SELECT id, first, last, age FROM Employees";
         ResultSet rs = stmt.executeQuery(sql);

         // 从结果集中提取数据
         while(rs.next()){
            // 根据列名称检索
            int id  = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last");

            // 显示值
            out.println("ID: " + id + "<br>");
            out.println(", Age: " + age + "<br>");
            out.println(", First: " + first + "<br>");
            out.println(", Last: " + last + "<br>");
         }
         out.println("</body></html>");

         // 清理环境
         rs.close();
         stmt.close();
```

```
            conn.close();
         }catch(SQLException se){
            // 处理 JDBC 错误
            se.printStackTrace();
         }catch(Exception e){
            // 处理 Class.forName 错误
            e.printStackTrace();
         }finally{
            // 最后是用于关闭资源的块
            try{
               if(stmt!=null)
                  stmt.close();
            }catch(SQLException se2){
            }// 我们不能做什么
            try{
               if(conn!=null)
               conn.close();
            }catch(SQLException se){
               se.printStackTrace();
            }//end finally try
         } //end try
      }
}
```

现在让我们来编译上面的 Servlet，并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
     <servlet-name>DatabaseAccess</servlet-name>
     <servlet-class>DatabaseAccess</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>DatabaseAccess</servlet-name>
     <url-pattern>/DatabaseAccess</url-pattern>
 </servlet-mapping>
....
```

现在调用这个 Servlet，输入链接：http://localhost:8080/DatabaseAccess，将显示以下响应结果：

```
<h1 align="center">数据库结果</h1>
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

# Servlet 文件上传

Servlet 可以与 HTML form 标签一起使用，来允许用户上传文件到服务器。上传的文件可以是文本文件或图像文件或任何文档。

## 创建一个文件上传表单

下面的 HTML 代码创建了一个文件上传表单。以下几点需要注意：

- 表单 **method** 属性应该设置为 **POST** 方法，不能使用 GET 方法。
- 表单 **enctype** 属性应该设置为 **multipart/form-data**.
- 表单 **action** 属性应该设置为在后端服务器上处理文件上传的 Servlet 文件。下面的实例使用了 **UploadServlet** Servlet 来上传文件。
- 上传单个文件，您应该使用单个带有属性 type="file" 的 <input .../> 标签。为了允许多个文件上传，请包含多个 name 属性值不同的 input 标签。输入标签具有不同的名称属性的值。浏览器会为每个 input 标签关联一个浏览按钮。

```
<html>
<head>
<title>文件上传表单</title>
</head>
<body>
<h3>文件上传：</h3>
请选择要上传的文件：<br />
<form action="UploadServlet" method="post"
                        enctype="multipart/form-data">
<input type="file" name="file" size="50" />
<br />
<input type="submit" value="上传文件" />
</form>
</body>
</html>
```

这将显示下面的结果，允许用户从本地计算机选择一个文件，当用户点击"上传文件"时，表单会连同从本地计算机选择的文件一起提交：

```
<b>文件上传：</b>
请选择要上传的文件：<br />
<input type="file" name="file" size="50" />
<br />
<input type="button" value="上传文件" />
<br />
注：这只是虚拟的表单，不会正常工作。
```

# 编写后台 Servlet

以下是 Servlet **UploadServlet**，会接受上传的文件，并把它储存在目录 <Tomcat-installation-directory>/webapps/data 中。这个目录名也可以使用外部配置来添加，比如 web.xml 中的 **context-param** 元素，如下所示：

```
<web-app>
....
<context-param>
    <description>Location to store uploaded file</description>
    <param-name>file-upload</param-name>
    <param-value>
         c:\apache-tomcat-5.5.29\webapps\data\
     </param-value>
</context-param>
....
</web-app>
```

以下是 UploadServlet 的源代码，可以一次处理多个文件的上传。在继续操作之前，请确认下列各项：

- 下面的实例依赖于 FileUpload，所以一定要确保在您的 classpath 中有最新版本的 **commons-fileupload.x.x.jar** 文件。可以从 http://commons.apache.org/fileupload/ 下载。
- FileUpload 依赖于 Commons IO，所以一定要确保在您的 classpath 中有最新版本的 **commons-io-x.x.jar** 文件。可以从 http://commons.apache.org/io/ 下载。
- 在测试下面实例时，您上传的文件大小不能大于 *maxFileSize*，否则文件将无法上传。
- 请确保已经提前创建好目录 c:\temp and c:\apache-tomcat-5.5.29\webapps\data。

```
// 导入必需的 java 库
import java.io.*;
import java.util.*;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.io.output.*;
```

```
public class UploadServlet extends HttpServlet {

   private boolean isMultipart;
   private String filePath;
   private int maxFileSize = 50 * 1024;
   private int maxMemSize = 4 * 1024;
   private File file ;

   public void init( ){
      // 获取文件将被存储的位置
      filePath =
             getServletContext().getInitParameter("file-upload");
   }
   public void doPost(HttpServletRequest request,
               HttpServletResponse response)
              throws ServletException, java.io.IOException {
      // 检查我们有一个文件上传请求
      isMultipart = ServletFileUpload.isMultipartContent(request);
      response.setContentType("text/html");
      java.io.PrintWriter out = response.getWriter( );
      if( !isMultipart ){
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet upload</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("<p>No file uploaded</p>");
         out.println("</body>");
         out.println("</html>");
         return;
      }
      DiskFileItemFactory factory = new DiskFileItemFactory();
      // 文件大小的最大值将被存储在内存中
      factory.setSizeThreshold(maxMemSize);
      // Location to save data that is larger than maxMemSize.
      factory.setRepository(new File("c:\\temp"));

      // 创建一个新的文件上传处理程序
      ServletFileUpload upload = new ServletFileUpload(factory);
      // 允许上传的文件大小的最大值
      upload.setSizeMax( maxFileSize );

      try{
      // 解析请求，获取文件项
      List fileItems = upload.parseRequest(request);

      // 处理上传的文件项
      Iterator i = fileItems.iterator();

      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet upload</title>");
      out.println("</head>");
```

```
        out.println("<body>");
        while ( i.hasNext () )
        {
           FileItem fi = (FileItem)i.next();
           if ( !fi.isFormField () )
           {
              // 获取上传文件的参数
              String fieldName = fi.getFieldName();
              String fileName = fi.getName();
              String contentType = fi.getContentType();
              boolean isInMemory = fi.isInMemory();
              long sizeInBytes = fi.getSize();
              // 写入文件
              if( fileName.lastIndexOf("\\") >= 0 ){
                 file = new File( filePath +
                 fileName.substring( fileName.lastIndexOf("\\"))) ;
              }else{
                 file = new File( filePath +
                 fileName.substring(fileName.lastIndexOf("\\")+1)) ;
              }
              fi.write( file ) ;
              out.println("Uploaded Filename: " + fileName + "<br>");
           }
        }
        out.println("</body>");
        out.println("</html>");
     }catch(Exception ex) {
         System.out.println(ex);
     }
     }
     public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
          throws ServletException, java.io.IOException {

          throw new ServletException("GET method used with " +
                  getClass( ).getName( )+": POST method required.");
     }
  }
```

## 编译和运行 Servlet

编译上面的 Servlet UploadServlet，并在 web.xml 文件中创建所需的条目，如下所示：

```
<servlet>
   <servlet-name>UploadServlet</servlet-name>
   <servlet-class>UploadServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>UploadServlet</servlet-name>
   <url-pattern>/UploadServlet</url-pattern>
</servlet-mapping>
```

现在尝试使用您在上面创建的 HTML 表单来上传文件。当您在浏览器中访问：http://localhost:8080/UploadFile.htm 时，它会显示下面的结果，这将有助于您从本地计算机上传任何文件。

```
<b>文件上传：</b>
请选择要上传的文件：<br />
<input type="file" name="file" size="50" />
<br />
<input type="button" value="上传文件" />
```

如果您的 Servelt 脚本能正常工作，那么您的文件会被上传到 c:\apache-tomcat-5.5.29\webapps\data\ 目录中。

# Servlet 处理日期

使用 Servlet 的最重要的优势之一是，可以使用核心 Java 中的大多数可用的方法。本章将讲解 Java 提供的 **java.util** 包中的 **Date** 类，这个类封装了当前的日期和时间。

Date 类支持两个构造函数。第一个构造函数初始化当前日期和时间的对象。

```
Date( )
```

下面的构造函数接受一个参数，该参数等于 1970 年 1 月 1 日午夜以来经过的毫秒数。

```
Date(long millisec)
```

一旦您有一个可用的 Date 对象，您可以调用下列任意支持的方法来使用日期：

| 方法 | 描述 |
|---|---|
| **boolean after(Date date)** | 如果调用的 Date 对象中包含的日期在 date 指定的日期之后，则返回 true，否则返回 false。 |
| **boolean before(Date date)** | 如果调用的 Date 对象中包含的日期在 date 指定的日期之前，则返回 true，否则返回 false。 |
| **Object clone( )** | 重复调用 Date 对象。 |
| **int compareTo(Date date)** | 把调用对象的值与 date 的值进行比较。如果两个值是相等的，则返回 0。如果调用对象在 date 之前，则返回一个负值。如果调用对象在 date 之后，则返回一个正值。 |
| **int compareTo(Object obj)** | 如果 obj 是 Date 类，则操作等同于 compareTo(Date)。否则，它会抛出一个 ClassCastException。 |
| **boolean equals(Object date)** | 如果调用的 Date 对象中包含的时间和日期与 date 指定的相同，则返回 true，否则返回 false。 |
| **long getTime( )** | 返回 1970 年 1 月 1 日以来经过的毫秒数。 |
| **int hashCode( )** | 为调用对象返回哈希代码。 |
| **void setTime(long time)** | 设置 time 指定的时间和日期，这表示从 1970 年 1 月 1 日午夜以来经过的时间（以毫秒为单位）。 |
| **String toString( )** | 转换调用的 Date 对象为一个字符串，并返回结果。 |

# 获取当前的日期和时间

在 Java Servlet 中获取当前的日期和时间是非常容易的。您可以使用一个简单的 Date 对象的 *toString()* 方法来输出当前的日期和时间，如下所示：

```
// 导入必需的 java 库
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class CurrentDate extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "显示当前的日期和时间";
      Date date = new Date();
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<h2 align=\"center\">" + date.toString() + "</h2>\n" +
        "</body></html>");
  }
}
```

现在，让我们来编译上面的 Servlet，并在 web.xml 文件中创建适当的条目，然后通过访问 http://localhost:8080/CurrentDate 来调用该 Servlet。这将会产生如下的结果：

```
<h1>显示当前的日期和时间</h1>

<h2>Mon Jun 21 21:46:49 GMT+04:00 2010</h2>
```

尝试刷新 URL http://localhost:8080/CurrentDate，每隔几秒刷新一次您都会发现显示时间的差异。

# 日期比较

正如上面所提到的，您可以在 Servlet 中使用所有可用的 Java 方法。如果您需要比较两个日期，以下是方法：

- 您可以使用 getTime() 来获取两个对象自 1970 年 1 月 1 日午夜以来经过的时间（以毫秒为单位），然后对这两个值进行比较。
- 您可以使用方法 before( )、after( ) 和 equals( )。由于一个月里 12 号在 18 号之前，例如，new Date(99, 2, 12).before(new Date (99, 2, 18)) 返回 true。
- 您可以使用 compareTo( ) 方法，该方法由 Comparable 接口定义，由 Date 实现。

# 使用 **SimpleDateFormat** 格式化日期

SimpleDateFormat 是一个以语言环境敏感的方式来格式化和解析日期的具体类。SimpleDateFormat 允许您选择任何用户定义的日期时间格式化的模式。

让我们修改上面的实例，如下所示：

```
// 导入必需的 java 库
import java.io.*;
import java.text.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class CurrentDate extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "显示当前的日期和时间";
      Date dNow = new Date( );
      SimpleDateFormat ft =
      new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<h2 align=\"center\">" + ft.format(dNow) + "</h2>\n" +
        "</body></html>");
  }
}
```

再次编译上面的 Servlet，然后通过访问 http://localhost:8080/CurrentDate 来调用该 Servlet。这将会产生如下的结果：

```
<h1>显示当前的日期和时间</h1>

<h2>Mon 2010.06.21 at 10:06:44 PM GMT+04:00</h2>
```

# 简单的日期格式的格式代码

使用事件模式字符串来指定时间格式。在这种模式下，所有的 ASCII 字母被保留为模式字母，这些字母定义如下：

| 字符 | 描述 | 实例 |
| --- | --- | --- |
| G | Era 指示器 | AD |
| y | 四位数表示的年 | 2001 |
| M | 一年中的月 | July 或 07 |
| d | 一月中的第几天 | 10 |
| h | 带有 A.M./P.M. 的小时（1~12） | 12 |
| H | 一天中的第几小时（0~23） | 22 |
| m | 一小时中的第几分 | 30 |
| s | 一分中的第几秒 | 55 |
| S | 毫秒 | 234 |
| E | 一周中的星期几 | Tuesday |
| D | 一年中的第几天 | 360 |
| F | 所在的周是这个月的第几周 | 2 (second Wed. in July) |
| w | 一年中的第几周 | 40 |
| W | 一月中的第几周 | 1 |
| a | A.M./P.M. 标记 | PM |
| k | 一天中的第几小时（1~24） | 24 |
| K | 带有 A.M./P.M. 的小时（0~11） | 10 |
| z | 时区 | Eastern Standard Time |
| ' | Escape for text | Delimiter |
| " | 单引号 | ` |

如需查看可用的处理日期方法的完整列表，您可以参考标准的 Java 文档。

# Servlet 网页重定向

当文档移动到新的位置，我们需要向客户端发送这个新位置时，我们需要用到网页重定向。当然，也可能是为了负载均衡，或者只是为了简单的随机，这些情况都有可能用到网页重定向。

重定向请求到另一个网页的最简单的方式是使用 response 对象的 **sendRedirect()** 方法。下面是该方法的定义：将请求重定向到另一页的最简单的方法是，用方法的 sendRedirect()的响应对象。以下是这种方法的定义：

```
public void HttpServletResponse.sendRedirect(String location)
throws IOException
```

该方法把响应连同状态码和新的网页位置发送回浏览器。您也可以通过把 setStatus() 和 setHeader() 方法一起使用来达到同样的效果：

```
....
String site = "http://www.newpage.com" ;
response.setStatus(response.SC_MOVED_TEMPORARILY);
response.setHeader("Location", site);
....
```

## 实例

本实例显示了 Servlet 如何进行页面重定向到另一个位置：

```
import java.io.*;
import java.sql.Date;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PageRedirect extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      // 要重定向的新位置
      String site = new String("http://www.w3cschool.cc");

      response.setStatus(response.SC_MOVED_TEMPORARILY);
      response.setHeader("Location", site);
    }
}
```

现在让我们来编译上面的 Servlet，并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
     <servlet-name>PageRedirect</servlet-name>
     <servlet-class>PageRedirect</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>PageRedirect</servlet-name>
     <url-pattern>/PageRedirect</url-pattern>
 </servlet-mapping>
....
```

现在通过访问 URL http://localhost:8080/PageRedirect 来调用这个 Servlet。这将把您转到给定的 URL http://www.w3cschool.cc 。

# Servlet 点击计数器

## 网页点击计数器

很多时候，您可能有兴趣知道网站的某个特定页面上的总点击量。使用 Servlet 来计算这些点击量是非常简单的，因为一个 Servlet 的生命周期是由它运行所在的容器控制的。

以下是实现一个简单的基于 Servlet 生命周期的网页点击计数器需要采取的步骤：

- 在 init() 方法中初始化一个全局变量。
- 每次调用 doGet() 或 doPost() 方法时，都增加全局变量。
- 如果需要，您可以使用一个数据库表来存储全局变量的值在 destroy() 中。在下次初始化 Servlet 时，该值可在 init() 方法内被读取。这一步是可选的。
- 如果您只想对一个 session 会话计数一次页面点击，那么请使用 isNew() 方法来检查该 session 会话是否已点击过相同页面。这一步是可选的。
- 您可以通过显示全局计数器的值，来在网站上展示页面的总点击量。这一步是可选的。

在这里，我们假设 Web 容器将无法重新启动。如果是重新启动或 Servlet 被销毁，计数器将被重置。

## 实例

本实例演示了如何实现一个简单的网页点击计数器：

```
import java.io.*;
import java.sql.Date;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PageHitCounter extends HttpServlet{

  private int hitCount;

  public void init()
  {
     // 重置点击计数器
     hitCount = 0;
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");
      // 该方法在 Servlet 被点击时执行
      // 增加 hitCount
      hitCount++;
      PrintWriter out = response.getWriter();
      String title = "总点击量";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<h2 align=\"center\">" + hitCount + "</h2>\n" +
        "</body></html>");

  }
  public void destroy()
  {
      // 这一步是可选的，但是如果需要，您可以把 hitCount 的值写入到数据库
  }
}
```

现在让我们来编译上面的 Servlet，并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
     <servlet-name>PageHitCounter</servlet-name>
     <servlet-class>PageHitCounter</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>PageHitCounter</servlet-name>
     <url-pattern>/PageHitCounter</url-pattern>
 </servlet-mapping>
....
```

现在通过访问 URL http://localhost:8080/PageHitCounter 来调用这个 Servlet。这将会在每次页面刷新时，把计数器的值增加 1，结果如下所示：

```
<h1>总点击量</h1>

<h2>6</h2>
```

# 网站点击计数器

很多时候，您可能有兴趣知道整个网站的总点击量。在 Servlet 中，这也是非常简单的，我们可以使用过滤器做到这一点。

以下是实现一个简单的基于过滤器生命周期的网站点击计数器需要采取的步骤：

- 在过滤器的 init() 方法中初始化一个全局变量。
- 每次调用 doFilter 方法时，都增加全局变量。
- 如果需要，您可以使用一个数据库表来存储全局变量的值在过滤器的 destroy() 中。在下次初始化过滤器时，该值可在 init() 方法内被读取。这一步是可选的。

在这里，我们假设 Web 容器将无法重新启动。如果是重新启动或 Servlet 被销毁，点击计数器将被重置。

# 实例

本实例演示了如何实现一个简单的网站点击计数器：

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class SiteHitCounter implements Filter{

  private int hitCount;

  public void  init(FilterConfig config)
                    throws ServletException{
     // 重置点击计数器
     hitCount = 0;
  }

  public void  doFilter(ServletRequest request,
              ServletResponse response,
              FilterChain chain)
              throws java.io.IOException, ServletException {

      // 把计数器的值增加 1
      hitCount++;

      // 输出计数器
      System.out.println("网站访问统计："+ hitCount );

      // 把请求传回到过滤器链
      chain.doFilter(request,response);
  }
  public void destroy()
  {
      // 这一步是可选的，但是如果需要，您可以把 hitCount 的值写入到数据库
  }
}
```

现在让我们来编译上面的 Servlet，并在 web.xml 文件中创建以下条目：

```
....
<filter>
   <filter-name>SiteHitCounter</filter-name>
   <filter-class>SiteHitCounter</filter-class>
</filter>

<filter-mapping>
   <filter-name>SiteHitCounter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

....
```

现在访问网站的任意页面，比如 http://localhost:8080/ 。这将会在每次任意页面被点击时，把计数器的值增加 1，它会在日志中显示以下消息：

```
网站访问统计： 1
网站访问统计： 2
网站访问统计： 3
网站访问统计： 4
网站访问统计： 5
..................
```

# Servlet 自动刷新页面

假设有一个网页，它是显示现场比赛成绩或股票市场状况或货币兑换率。对于所有这些类型的页面，您需要定期刷新网页。

Java Servlet 提供了一个机制，使得网页会在给定的时间间隔自动刷新。

刷新网页的最简单的方式是使用响应对象的方法 **setIntHeader()**。以下是这种方法的定义：

```
public void setIntHeader(String header, int headerValue)
```

此方法把头信息 "Refresh" 连同一个表示时间间隔的整数值（以秒为单位）发送回浏览器。

## 自动刷新页面实例

本实例演示了 Servlet 如何使用 **setIntHeader()** 方法来设置 **Refresh** 头信息，从而实现自动刷新页面。

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// 扩展 HttpServlet 类
public class Refresh extends HttpServlet {

  // 处理 GET 方法请求的方法
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置刷新自动加载的事件间隔为 5 秒
      response.setIntHeader("Refresh", 5);

      // 设置响应内容类型
      response.setContentType("text/html");

      // 获取当前的时间
      Calendar calendar = new GregorianCalendar();
      String am_pm;
      int hour = calendar.get(Calendar.HOUR);
      int minute = calendar.get(Calendar.MINUTE);
      int second = calendar.get(Calendar.SECOND);
```

```
      if(calendar.get(Calendar.AM_PM) == 0)
        am_pm = "AM";
      else
        am_pm = "PM";

      String CT = hour+":"+ minute +":"+ second +" "+ am_pm;

      PrintWriter out = response.getWriter();
      String title = "使用 Servlet 自动刷新页面";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n"+
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<p>当前时间是：" + CT + "</p>\n");
  }
  // 处理 POST 方法请求的方法
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
  }
}
```

现在让我们来编译上面的 Servlet，并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
     <servlet-name>Refresh</servlet-name>
     <servlet-class>Refresh</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>Refresh</servlet-name>
     <url-pattern>/Refresh</url-pattern>
 </servlet-mapping>
....
```

现在通过访问 URL http://localhost:8080/Refresh 来调用这个 Servlet。这将会每隔 5 秒钟显示一次当前系统时间。运行该 Servlet，并等待查看结果：

```
<h1>使用 Servlet 自动刷新页面</h1>

当前时间是：9:44:50 PM
```

# Servlet 发送电子邮件

使用 Servlet 发送一封电子邮件是很简单的，但首先您必须在您的计算机上安装 **JavaMail API 和 Java Activation Framework）JAF）**。

- 您可以从 Java 标准网站下载最新版本的 JavaMail（版本 1.2）。
- 您可以从 Java 标准网站下载最新版本的 JAF（版本 1.1.1）。

下载并解压缩这些文件，在新创建的顶层目录中，您会发现这两个应用程序的一些 jar 文件。您需要把 **mail.jar** 和 **activation.jar** 文件添加到您的 CLASSPATH 中。

## 发送一封简单的电子邮件

下面的实例将从您的计算机上发送一封简单的电子邮件。这里假设您的本地主机已连接到互联网，并支持发送电子邮件。同时确保 Java Email API 包和 JAF 包的所有的 jar 文件在 CLASSPATH 中都是可用的。

```
// 文件名 SendEmail.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendEmail extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 收件人的电子邮件 ID
      String to = "abcd@gmail.com";

      // 发件人的电子邮件 ID
      String from = "web@gmail.com";

      // 假设您是从本地主机发送电子邮件
      String host = "localhost";

      // 获取系统的属性
      Properties properties = System.getProperties();

      // 设置邮件服务器
      properties.setProperty("mail.smtp.host", host);
```

```
        // 获取默认的 Session 对象
        Session session = Session.getDefaultInstance(properties);

        // 设置响应内容类型
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        try{
           // 创建一个默认的 MimeMessage 对象
           MimeMessage message = new MimeMessage(session);
           // 设置 From: header field of the header.
           message.setFrom(new InternetAddress(from));
           // 设置 To: header field of the header.
           message.addRecipient(Message.RecipientType.TO,
                                new InternetAddress(to));
           // 设置 Subject: header field
           message.setSubject("This is the Subject Line!");
           // 现在设置实际消息
           message.setText("This is actual message");
           // 发送消息
           Transport.send(message);
           String title = "发送电子邮件";
           String res = "成功发送消息...";
           String docType =
           "<!doctype html public \"-//w3c//dtd html 4.0 " +
           "transitional//en\">\n";
           out.println(docType +
           "<html>\n" +
           "<head><title>" + title + "</title></head>\n" +
           "<body bgcolor=\"#f0f0f0\">\n" +
           "<h1 align=\"center\">" + title + "</h1>\n" +
           "<p align=\"center\">" + res + "</p>\n" +
           "</body></html>");
        }catch (MessagingException mex) {
           mex.printStackTrace();
        }
     }
  }
```

现在让我们来编译上面的 Servlet，并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
     <servlet-name>SendEmail</servlet-name>
     <servlet-class>SendEmail</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>SendEmail</servlet-name>
     <url-pattern>/SendEmail</url-pattern>
 </servlet-mapping>
....
```

现在通过访问 URL http://localhost:8080/SendEmail 来调用这个 Servlet。这将会发送一封电子邮件到给定的电子邮件 ID *abcd@gmail.com*，并将显示下面所示的响应：

```
<h1>发送电子邮件</h1>

成功发送消息...
```

如果您想要发送一封电子邮件给多个收件人，那么请使用下面的方法来指定多个电子邮件 ID：

```
void addRecipients(Message.RecipientType type,
                   Address[] addresses)
throws MessagingException
```

下面是对参数的描述：

- **type**：这将被设置为 TO、CC 或 BCC。在这里，CC 代表抄送，BCC 代表密件抄送。例如 *Message.RecipientType.TO*。
- **addresses**：这是电子邮件 ID 的数组。当指定电子邮件 ID 时，您需要使用 InternetAddress() 方法。

# 发送一封 HTML 电子邮件

下面的实例将从您的计算机上发送一封 HTML 格式的电子邮件。这里假设您的本地主机已连接到互联网，并支持发送电子邮件。同时确保 Java Email API 包和 JAF 包的所有的 jar 文件在 CLASSPATH 中都是可用的。

本实例与上一个实例很类似，但是这里我们使用 setContent() 方法来设置第二个参数为 "text/html" 的内容，该参数用来指定 HTML 内容是包含在消息中的。

使用这个实例，您可以发送内容大小不限的 HTML 内容。

```
// 文件名 SendEmail.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendEmail extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 收件人的电子邮件 ID
      String to = "abcd@gmail.com";

      // 发件人的电子邮件 ID
      String from = "web@gmail.com";

      // 假设您是从本地主机发送电子邮件
      String host = "localhost";

      // 获取系统的属性
      Properties properties = System.getProperties();

      // 设置邮件服务器
      properties.setProperty("mail.smtp.host", host);

      // 获取默认的 Session 对象
      Session session = Session.getDefaultInstance(properties);

      // 设置响应内容类型
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      try{
         // 创建一个默认的 MimeMessage 对象
         MimeMessage message = new MimeMessage(session);
         // 设置 From: header field of the header.
         message.setFrom(new InternetAddress(from));
         // 设置 To: header field of the header.
         message.addRecipient(Message.RecipientType.TO,
                                  new InternetAddress(to));
         // 设置 Subject: header field
         message.setSubject("This is the Subject Line!");

         // 设置实际的 HTML 消息，内容大小不限
         message.setContent("<h1>This is actual message</h1>",
                            "text/html" );
         // 发送消息
```

```
         Transport.send(message);
         String title = "发送电子邮件";
         String res = "成功发送消息...";
         String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";
         out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor=\"#f0f0f0\">\n" +
         "<h1 align=\"center\">" + title + "</h1>\n" +
         "<p align=\"center\">" + res + "</p>\n" +
         "</body></html>");
      }catch (MessagingException mex) {
         mex.printStackTrace();
      }
   }
}
```

编译并运行上面的 Servlet，在给定的电子邮件 ID 上发送 HTML 消息。

# 在电子邮件中发送附件

下面的实例将从您的计算机上发送一封带有附件的电子邮件。这里假设您的本地主机已连接到互联网，并支持发送电子邮件。同时确保 Java Email API 包和 JAF 包的所有的 jar 文件在 CLASSPATH 中都是可用的。

```
// 文件名 SendEmail.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendEmail extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 收件人的电子邮件 ID
      String to = "abcd@gmail.com";

      // 发件人的电子邮件 ID
      String from = "web@gmail.com";

      // 假设您是从本地主机发送电子邮件
      String host = "localhost";
```

```
      // 获取系统的属性
      Properties properties = System.getProperties();

      // 设置邮件服务器
      properties.setProperty("mail.smtp.host", host);

      // 获取默认的 Session 对象
      Session session = Session.getDefaultInstance(properties);

      // 设置响应内容类型
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

       try{
         // 创建一个默认的 MimeMessage 对象
         MimeMessage message = new MimeMessage(session);

         // 设置 From: header field of the header.
         message.setFrom(new InternetAddress(from));

         // 设置 To: header field of the header.
         message.addRecipient(Message.RecipientType.TO,
                                  new InternetAddress(to));

         // 设置 Subject: header field
         message.setSubject("This is the Subject Line!");

         // 创建消息部分
         BodyPart messageBodyPart = new MimeBodyPart();

         // 填写消息
         messageBodyPart.setText("This is message body");

         // 创建一个多部分消息
         Multipart multipart = new MimeMultipart();

         // 设置文本消息部分
         multipart.addBodyPart(messageBodyPart);

         // 第二部分是附件
         messageBodyPart = new MimeBodyPart();
         String filename = "file.txt";
         DataSource source = new FileDataSource(filename);
         messageBodyPart.setDataHandler(new DataHandler(source));
         messageBodyPart.setFileName(filename);
         multipart.addBodyPart(messageBodyPart);

         // 发送完整的消息部分
         message.setContent(multipart );

         // 发送消息
         Transport.send(message);
```

```
         String title = "发送电子邮件";
         String res = "成功发送电子邮件...";
         String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";
         out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor=\"#f0f0f0\">\n" +
         "<h1 align=\"center\">" + title + "</h1>\n" +
         "<p align=\"center\">" + res + "</p>\n" +
         "</body></html>");
      }catch (MessagingException mex) {
         mex.printStackTrace();
      }
   }
}
```

编译并运行上面的 Servlet，在给定的电子邮件 ID 上发送带有文件附件的消息。

## 用户身份认证部分

如果需要向电子邮件服务器提供用户 ID 和密码进行身份认证，那么您可以设置如下属性：

```
props.setProperty("mail.user", "myuser");
props.setProperty("mail.password", "mypwd");
```

电子邮件发送机制的其余部分与上面讲解的保持一致。

# Servlet 包

涉及到 WEB-INF 子目录的 Web 应用程序结构是所有的 Java web 应用程序的标准，并由 Servlet API 规范指定。给定一个顶级目录名 myapp，目录结构如下所示：

```
/myapp
    /images
    /WEB-INF
        /classes
        /lib
```

WEB-INF 子目录中包含应用程序的部署描述符，名为 web.xml。所有的 HTML 文件都位于顶级目录 *myapp* 下。对于 admin 用户，您会发现 ROOT 目录是 myApp 的父目录。

## 创建包中的 Servlet

WEB-INF/classes 目录包含了所有的 Servlet 类和其他类文件，类文件所在的目录结构与他们的包名称匹配。例如，如果您有一个完全合格的类名称 **com.myorg.MyServlet**，那么这个 Servlet 类必须位于以下目录中：

```
/myapp/WEB-INF/classes/com/myorg/MyServlet.class
```

下面的例子创建包名为 *com.myorg* 的 MyServlet 类。

```
// 为包命名
package com.myorg;

// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

  private String message;

  public void init() throws ServletException
  {
      // 执行必需的的初始化
      message = "Hello World";
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 设置响应内容类型
      response.setContentType("text/html");

      // 实际的逻辑是在这里
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
  }

  public void destroy()
  {
      // 什么也不做
  }
}
```

# 编译包中的 Servlet

编译包中的类与编译其他的类没有什么大的不同。最简单的方法是让您的 java 文件保留完全限定路径，如上面提到的类，将被保留在 com.myorg 中。您还需要在 CLASSPATH 中添加该目录。

假设您的环境已正确设置，进入 **<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes** 目录，并编译 MyServlet.java，如下所示：

```
$ javac MyServlet.java
```

如果 Servlet 依赖于其他库，那么您必须在 CLASSPATH 中也要引用那些 JAR 文件。这里我只引用了 servlet-api.jar JAR 文件，因为我在 Hello World 程序中并没有使用任何其他库。

该命令行使用内置的 javac 编译器，它是 Sun Microsystems Java 软件开发工具包（JDK，全称 Java Software Development Kit）附带的。 Microsystems的Java软件开发工具包（JDK）。为了让该命令正常工作，必须包括您在 PATH 环境变量中所使用的 Java SDK 的位置。

如果一切顺利，上述编译会在同一目录下生成 **MyServlet.class** 文件。下一节将解释如何部把一个已编译的 Servlet 部署到生产中。

## Servlet 打包部署

默认情况下，Servlet 应用程序位于路径 <Tomcat-installation-directory>/webapps/ROOT 下，且类文件放在 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中。

如果您有一个完全合格的类名称 **com.myorg.MyServlet**，那么这个 Servlet 类必须位于 WEB-INF/classes/com/myorg/MyServlet.class 中，您需要在位于 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/ 的 web.xml 文件中创建以下条目：

```
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>com.myorg.MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
```

上面的条目要被创建在 web.xml 文件中的 <web-app>...</web-app> 标签内。在该文件中可能已经有各种可用的条目，但不要在意。

到这里，您基本上已经完成了，现在让我们使用 <Tomcat-installation-directory>\bin\startup.bat（在 Windows 上）或 <Tomcat-installation-directory>/bin/startup.sh（在 Linux/Solaris 等上）启动 tomcat 服务器，最后在浏览器的地址栏中输入 **http://localhost:8080/MyServlet**。如果一切顺利，您会看到下面的结果：

```
<h1>Hello World</h1>
```

# Servlet 调试

测试/调试 Servlet 始终是开发使用过程中的难点。Servlet 往往涉及大量的客户端/服务器交互，可能会出现错误但又难以重现。

这里有一些提示和建议，可以帮助您调试。

## System.out.println()

System.out.println() 是作为一个标记来使用的，用来测试一段特定的代码是否被执行。我们也可以打印出变量的值。此外：

- 由于 System 对象是核心 Java 对象的一部分，它可以在不需要安装任何额外类的情况下被用于任何地方。这包括 Servlet、JSP、RMI、EJB's、普通的 Beans 和类，以及独立的应用程序。
- 与在断点处停止不同，写入到 System.out 不会干扰到应用程序的正常执行流程，这使得它在时序是至关重要的时候显得尤为有价值。

下面是使用 System.out.println() 的语法：

```
System.out.println("Debugging message");
```

通过上面的语法生成的所有消息将被记录在 Web 服务器日志文件中。

## 消息日志

使用适当的日志记录方法来记录所有调试、警告和错误消息，这是非常好的想法，推荐使用 log4J 来记录所有的消息。

Servlet API 还提供了一个简单的输出信息的方式，使用 log() 方法，如下所示：

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ContextLog extends HttpServlet {
  public void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
         java.io.IOException {

      String par = request.getParameter("par1");
      // 调用两个 ServletContext.log 方法
      ServletContext context = getServletContext( );

      if (par == null || par.equals(""))
      // 通过 Throwable 参数记录版本
      context.log("No message received:",
          new IllegalStateException("Missing parameter"));
      else
          context.log("Here is the visitor's message: " + par);

      response.setContentType("text/html");
      java.io.PrintWriter out = response.getWriter( );
      String title = "Context Log";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<h2 align=\"center\">Messages sent</h2>\n" +
        "</body></html>");
    } //doGet
}
```

ServletContext 把它的文本消息记录到 Servlet 容器的日志文件中。对于 Tomcat，这些日志可以在 <Tomcat-installation-directory>/logs 目录中找到。

这些日志文件确实对新出现的错误或问题的频率给出指示。正因为如此，建议在通常不会发生的异常的 catch 子句中使用 log() 函数。

## 使用 JDB 调试器

您可以使用调试 applet 或应用程序的 jdb 命令来调试 Servlet。

为了调试一个 Servlet，我们可以调试 sun.servlet.http.HttpServer，然后把它看成是 HttpServer 执行 Servlet 来响应浏览器端的 HTTP 请求。这与调试 applet 小程序非常相似。与调试 applet 不同的是，实际被调试的程序是 sun.applet.AppletViewer。

大多数调试器会自动隐藏如何调试 applet 的细节。同样的，对于 servlet，您必须帮调试器执行以下操作：

- 设置您的调试器的类路径 classpath，以便它可以找到 sun.servlet.http.Http-Server 和相关的类。
- 设置您的调试器的类路径 classpath，以便它可以找到您的 servlet 和支持的类，通常是在 server_root/servlets 和 server_root/classes 中。

您通常不会希望 server_root/servlets 在您的 classpath 中，因为它会禁用 servlet 的重新加载。但是这种包含规则对于调试是非常有用的。它允许您的调试器在 HttpServer 中的自定义 Servlet 加载器加载 Servlet 之前在 Servlet 中设置断点。

如果您已经设置了正确的类路径 classpath，就可以开始调试 sun.servlet.http.HttpServer。可以在您想要调试的 Servlet 代码中设置断点，然后通过 Web 浏览器使用给定的 Servlet（http://localhost:8080/servlet/ServletToDebug）向 HttpServer 发出请求。您会看到程序执行到断点处会停止。

## 使用注释

代码中的注释有助于以各种方式进行调试。注释可用于调试过程的很多其他方式中。

该 Servlet 使用 Java 注释和单行注释（//...），多行注释（/ .../）可用于暂时移除部分 Java 代码。如果 bug 消失，仔细看看您刚才注释的代码并找出问题所在。

## 客户端和服务器端头信息

有时，当一个 Servlet 并没有像预期那样时，查看原始的 HTTP 请求和响应是非常有用的。如果您熟悉 HTTP 结构，您可以阅读请求和响应，看看这些头信息究竟是什么。

## 重要的调试技巧

下面列出了一些 Servlet 调试的技巧：

- 请注意，server_root/classes 不会重载，而 server_root/servlets 可能会。
- 要求浏览器显示它所显示的页面的原始内容。这有助于识别格式的问题。它通常是"视图"菜单下的一个选项。
- 通过强制执行完全重新加载页面来确保浏览器还没有缓存前一个请求的输出。在 Netscape Navigator 中，请使用 Shift-Reload，在 Internet Explorer 中，请使用 Shift-Refresh。

- 请确认 servlet 的 init() 方法接受一个 ServletConfig 参数，并调用 super.init(config)。

# Servlet 国际化

在我们开始之前，先来看看三个重要术语：

- 国际化（**i18n**）：这意味着一个网站提供了不同版本的翻译成访问者的语言或国籍的内容。
- 本地化（**l10n**）：这意味着向网站添加资源，以使其适应特定的地理或文化区域，例如网站翻译成印地文（Hindi）。
- 区域设置（**locale**）：这是一个特殊的文化或地理区域。它通常指语言符号后跟一个下划线和一个国家符号。例如 "en_US" 表示针对 US 的英语区域设置。

当建立一个全球性的网站时有一些注意事项。本教程不会讲解这些注意事项的完整细节，但它会通过一个很好的实例向您演示如何通过差异化定位（即区域设置）来让网页以不同语言呈现。

Servlet 可以根据请求者的区域设置拾取相应版本的网站，并根据当地的语言、文化和需求提供相应的网站版本。以下是 request 对象中返回 Locale 对象的方法。

```
java.util.Locale request.getLocale()
```

## 检测区域设置

下面列出了重要的区域设置方法，您可以使用它们来检测请求者的地理位置、语言和区域设置。下面所有的方法都显示了请求者浏览器中设置的国家名称和语言名称。

| 方法 | 描述 |
|---|---|
| **String getCountry()** | 该方法以 2 个大写字母形式的 ISO 3166 格式返回该区域设置的国家/地区代码。 |
| **String getDisplayCountry()** | 该方法返回适合向用户显示的区域设置的国家的名称。 |
| **String getLanguage()** | 该方法以小写字母形式的 ISO 639 格式返回该区域设置的语言代码。 |
| **String getDisplayLanguage()** | 该方法返回适合向用户显示的区域设置的语言的名称。 |
| **String getISO3Country()** | 该方法返回该区域设置的国家的三个字母缩写。 |
| **String getISO3Language()** | 该方法返回该区域设置的语言的三个字母的缩写。 |

# 实例

本实例演示了如何显示某个请求的语言和相关的国家：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;

public class GetLocale extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // 获取客户端的区域设置
      Locale locale = request.getLocale();
      String language = locale.getLanguage();
      String country = locale.getCountry();

      // 设置响应内容类型
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String title = "检测区域设置";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + language + "</h1>\n" +
        "<h2 align=\"center\">" + country + "</h2>\n" +
        "</body></html>");
  }
}
```

# 语言设置

Servlet 可以输出以西欧语言（如英语、西班牙语、德语、法语、意大利语、荷兰语等）编写的页面。在这里，为了能正确显示所有的字符，设置 Content-Language 头是非常重要的。

第二点是使用 HTML 实体显示所有的特殊字符，例如，"ñ" 表示 "?"，"¡" 表示 "?"，如下所示：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;

public class DisplaySpanish extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // 设置响应内容类型
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // 设置西班牙语言代码
    response.setHeader("Content-Language", "es");

    String title = "En Espa&ntilde;ol";
    String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
     "<html>\n" +
     "<head><title>" + title + "</title></head>\n" +
     "<body bgcolor=\"#f0f0f0\">\n" +
     "<h1>" + "En Espa&ntilde;ol:" + "</h1>\n" +
     "<h1>" + "&iexcl;Hola Mundo!" + "</h1>\n" +
     "</body></html>");
  }
}
```

# 特定于区域设置的日期

您可以使用 java.text.DateFormat 类及其静态方法 getDateTimeInstance() 来格式化特定于区域设置的日期和时间。下面的实例演示了如何格式化特定于某个给定的区域设置的日期：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
import java.text.DateFormat;
import java.util.Date;

public class DateLocale extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // 设置响应内容类型
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // 获取客户端的区域设置
    Locale locale = request.getLocale( );
    String date = DateFormat.getDateTimeInstance(
                                  DateFormat.FULL,
                                  DateFormat.SHORT,
                                  locale).format(new Date( ));

    String title = "特定于区域设置的日期";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + date + "</h1>\n" +
      "</body></html>");
  }
}
```

## 特定于区域设置的货币

您可以使用 java.text.NumberFormat 类及其静态方法 getCurrencyInstance() 来格式化数字（比如 long 类型或 double 类型）为特定于区域设置的货币。下面的实例演示了如何格式化特定于某个给定的区域设置的货币：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
import java.text.NumberFormat;
import java.util.Date;

public class CurrencyLocale extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // 设置响应内容类型
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // 获取客户端的区域设置
    Locale locale = request.getLocale( );
    NumberFormat nft = NumberFormat.getCurrencyInstance(locale);
    String formattedCurr = nft.format(1000000);

    String title = "特定于区域设置的货币";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + formattedCurr + "</h1>\n" +
      "</body></html>");
  }
}
```

# 特定于区域设置的百分比

您可以使用 java.text.NumberFormat 类及其静态方法 getPercentInstance() 来格式化特定于区域设置的百分比。下面的实例演示了如何格式化特定于某个给定的区域设置的百分比：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
import java.text.NumberFormat;
import java.util.Date;

public class PercentageLocale extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // 设置响应内容类型
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // 获取客户端的区域设置
    Locale locale = request.getLocale( );
    NumberFormat nft = NumberFormat.getPercentInstance(locale);
    String formattedPerc = nft.format(0.51);

    String title = "特定于区域设置的百分比";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + formattedPerc + "</h1>\n" +
      "</body></html>");
  }
}
```

# Spring教程

Spring框架，由Rod Johnson开发，是一个非常强大的反转控制（IOC）框架，以帮助分离项目组件的依赖关系。在本系列教程，将多个步骤提供一些例子，用来学习和解释Spring框架。新的 Spring 4.1 教程 (2015/12/12) 增加了关于使用Spring EL，JavaConfig，AspectJ和Spring对象/XML映射（OXM）等许多Spring 4.0 的实例教程。

## Spring快速入门

快速入门了解Spring框架开发的基础。

- 1.0、Spring hello world实例 Spring3.0的hello world 实例，在新的Spring3.0开发需要什么。
- 1.1、Spring松耦合的实例一个例子用来演示Spring是如何使组件之间松散耦合的。

## Spring JavaConfig

Spring 3.0支持JavaConfig，现在可以使用注解来配置Spring。

- 1.2、Spring JavaConfig 实例 使用@Configuration和@Bean演示到在Spring定义bean
- 1.3、Spring JavaConfig @Import 实例 使用@Import组织Bean类模块化的实例

## Spring 依赖注入(DI)

Spring如何使用依赖注入（DO）来管理对象的依赖关系。

- 1.4、Spring依赖注入 (DI) Spring如何通过应用Setter注入和构造器注入的依赖注入（DI）设计模式实例
- 1.5、Spring DI 通过setter方法 通过setter方法依赖注入bean
- 1.6、Spring DI 构造方法 可以通过构造方法依赖注入bean
- 1.7.1、Spring构造方法注入类型歧义 构造函数注入参数类型歧义的问题总是发生在一个bean包含多个构造方法有多个参数

## Bean基础

只需要在Spring IoC容器使用的类被认为是“Bean”，并可在Spring bean 的配置文件或者通过注解来声明。

- 1.7、Spring bean 引用实例 如何指定相同或不同的bean配置文件中的bean引用互相访问。
- 1.8、注入值到Spring Bean的属性 三种方法注入值到bean的属性。
- 1.9、加载多个Spring bean配置文件 开发人员总是在不同的模块文件夹归类不同的 bean 配置文件，这里有一个技巧，向您展示如何加载多个Spring bean的配置文件。
- 2.0、Spring内部Bean实例 每当一个bean仅用于一个特定的属性，它总是建议将其声明为一个内部bean。
- 2.1、Spring bean作用域 Bean作用域是用来决定哪些bean实例的类型应该是从Spring容器中返回给调用者。
- 2.2、Spring集合 (List, Set, Map, and Properties) 实例 注入值到集合类型（列表，集，映射和属性）实例。
- 2.3、ListFactoryBean实例 创建一个具体的列表集合类（ArrayList 和 LinkedList），并注入到 bean 属性。
- 2.4、SetFactoryBean实例 创建一组具体的Set集合类（HashSet和TreeSet），并注入到bean属性。
- 2.5、MapFactoryBean实例 创建一个具体的映射集合类（HashMap和TreeMap中），并注入到bean属性。
- 2.6、Spring注入日期到bean属性 – CustomDateEditor 一般情况下，Spring接受日期变量，这里有一个技巧 - 使用CustomDateEditor来解决。
- 2.7、Spring PropertyPlaceholderConfigurer实例 通过一个特殊格式具体化部署详情到一个属性文件，并从bean配置文件访问获得 – ${variable}
- 2.8、Spring bean配置继承 继承是非常有用，一个bean来分享共同的值，属性或配置。
- 2.9、Spring依赖检查 Spring4中提供了依赖检查模式，以确保所要求的属性已经在bean中设置。
- 3.0、Spring使用@Required注解依赖检查 依赖检查注释模式。
- 3.1、自定义@Required-style注解 创建自定义@Required-style注解，相当于@Required 注解
- 3.2、Bean InitializingBean 和 DisposableBean实例 执行 bean 初始化和销毁某些动作（接口）
- 3.3、Bean init-method 和 destroy-method 实例 执行 bean 初始化和销毁某些动作（XML）
- 3.4、Bean @PostConstruct 和 @PreDestroy 实例 执行 bean 初始化和销毁某些动作（注解）

## Spring表达式语言

Spring 3.0引入了Spring表达语言，或Spring EL丰富而强大的表达式语言。

- 3.5、Spring EL hello world 实例 使用 Spring 表达式语言（EL）快速入门
- 3.6、Spring EL bean 引用实例 参考bean，bean属性使用一个点（.）符号。
- 3.7、Spring EL 方法调用实例 直接调用Bean方法
- 3.8、Spring EL 操作符实例 Spring EL支持大多数标准的关系，逻辑和数学运算符。
- 3.9、Spring EL 三元操作符(if-then-else) 实例 有条件的检查：if else then.
- 4.0、Spring EL Arrays, Lists, Maps 实例 适用于映射和列表。

- 4.1、Spring EL 正则表达式实例 正则表达式来计算评估条件。
- 4.2、 测试 Spring EL 与 ExpressionParser 如何轻松地测试Spring El。

## Spring自动组件扫描

Spring是能够扫描，检测并自动注册 bean 的。

- 4.3、Spring自动扫描组件 让Spring自动扫描，检测和注册Bean。
- 4.4、Spring过滤器组件自动扫描 过滤器自动扫描模式的某些组件实例。

## Spring自动装配Bean

Spring自动装配“auto-wiring"来装配或Bean类，无论是在XML和注释。

- 4.5、Spring自动装配Beans Spring 5种自动装配方式总结。
- 4.6、Spring由类型(Type)自动装配 如果一个 bean 的数据类型是用其它 bean 属性的数据类型，那么自动装配它。
- 4.7、Spring由名称(Name)自动装配 如果一个 bean 的名称与其他bean属性的名称是一样的，那么可以自动装配它。
- 4.8、Spring由构造方法自动装配 由在构造函数的参数类型自动装配。
- 4.9、Spring由AutoDetect自动装配 这意味着选择“用构造函数自动装配”，如果默认构造函数被找到，否则使用“自动装配按类型”。
- 5.0、Spring 使用@Autowired注解自动装配 例子来说明如何定义注解 “auto-wiring” 模式。
- 5.1、Spring 自动装配@Qualifier实例 举个例子来确定哪些Bean有资格自动装配到某个字段上。

## Spring AOP (面向方面编程)

Spring AOP的模块化方面横切关注点。简单地说，就是一个拦截器拦截一些方法。

- 5.2、Spring AOP通知实例 – Advice 有关不同类型的 Spring 建议说明和示例。
- 5.3、Spring AOP实例 – Pointcut , Advisor 有关不同类型的 Spring 的切入点和 Advisor 的解释和示例。
- 5.4、Spring AOP拦截器的序列 AOP拦截器的顺序会影响功能。
- 5.5、 自动代理创建者实例 一个自动代理生成例子是用来为Bean自动创建代理对象，以避免造成许多重复的代理对象。

## Spring AOP + AspectJ框架

由于Spring 2.0中对 AspectJ 更加灵活和强大的支持。然而，本实例在 Spring3.0 中应用说明。

- 5.6、Spring AOP + AspectJ注解实例 一个例子向您展示如何将AspectJ 注解

与Spring框架集成。
- 5.7、Spring AOP + AspectJ 在 XML配置实例 Spring AOP 使用AspectJ 在XML基本配置。

## Spring Object/XML 映射器

在Spring3.0，对象到XML映射（OXM）从Spring Web服务到核心Spring框架。

- 5.8、Spring Object/XML映射实例 Spring oxm + castor，转换对象到XML，反之亦然。

## Spring JDBC支持

Spring提供了很多辅助类简化整个 JDBC 数据库操作。

- 5.9、Spring + JDBC实例 一个例子来说明如何集成 Spring 和 JDBC。
- 6.0、JdbcTemplate + JdbcDaoSupport 实例 使用Spring的JdbcTemplate和JdbcDaoSupport 类来简化整个JDBC数据库操作过程的一个实例。
- 6.1、Spring JdbcTemplate查询实例 这里有一些例子来说明如何使用JdbcTemplate的query() 方法来查询或从数据库中提取数据。
- 6.2、Spring JdbcTemplate batchUpdate() 实例 这里是一个BATCHUPDATE（）例子来说明如何执行批量插入操作。
- 6.3、Spring SimpleJdbcTemplate查询示例 更多的用户查询操作，或从数据库中提取数据的友好而简单的方法。
- 6.4、Spring SimpleJdbcTemplate batchUpdate()实例 使用SimpleJdbcTemplate类，一个Java5 的友好补充 JdbcTemplate 的一个批量更新的例子。
- 6.5、Spring SimpleJdbcTemplate类命名参数实例一个例子来说明如何使用命名参数作为SQL参数值，而这仅仅是在SimpleJdbcTemplate中支持。

## Spring Hibernate支持

Spring 提供了许多方便的类来支持 Hibernate ORM框架。

- 6.6、Spring+Hibernate+MySql实例 使用 Spring 和 Hibernate 的一个简单的项目。
- 6.7、Spring + Hibernate+ MySql注解实例 使用Spring和Hibernate（注释版）的一个简单的项目。
- 6.8、Spring AOP在Hibernate的事务管理 一个例子来说明如何管理Hibernate事务与Spring AOP。
- Struts + Spring + Hibernate集成 Spring、Struts和Hibernate框架集成的实例。

## Spring E-mail支持

Spring 提供 MailSender 通过 JavaMail API 发送电子邮件。

- 6.9、通过MailSender发送电子邮件 使用Spring的MailSender通过Gmail的SMTP服务器发送电子邮件实例。
- 7.0、在bean配置文件中的电子邮件模板硬编码所有电子邮件属性和消息，正文内容不是一个好的做法，应该考虑在 Spring的bean配置文件中定义电子邮件模板。
- 7.1、Spring发送带附件的Email 使用Spring来发送带附件的电子邮件示例。

## Spring与其它Web框架集成

Spring集成其它Web框架。

- 7.2、Spring依赖注入servlet会话监听器Spring提供了一个"ContextLoaderListener"监听器作为一个通用的方法，以使Spring依赖注入会话监听器应用到几乎所有其他的web框架。
- Struts 2 + Spring集成实例 Spring和Struts2框架集成的实例。
- Struts 2 + Spring + Quartz 计划程序集成示例 Spring + Struts 2 + Quartz集成实例
- Struts 2 + Spring + Hibernate集成实例 Spring+Struts2+Hibernate的集成实例。

## Spring问答

- 7.3、= 资源捆绑ResourceBundleMessageSource示例 ResourceBundleMessageSource解决文本信息在不同地区的最常见类。
- 7.4、访问MessageSource的bean（MessageSourceAware）一个例子来说明如何经由MessageSourceAware 接口的 bean 得到 MessageSource。
- 资源加载程序的getResource()示例Spring资源加载器提供了一个非常通用的getResource（）方法来从文件系统，类路径或URL得到这样的资源（文本文件，媒体文件，图像文件...）。

## Spring常见错误

一些 Spring 常见的错误消息：

- Cannot proxy target class because CGLIB2 is not available
- CGLIB is required to process @Configuration classes
- java.lang.ClassNotFoundException: org.exolab.castor.xml.XMLException
- java.lang.ClassNotFoundException: org.apache.xml.serialize.XMLSerializer

## Spring参考

- Spring Framework (Wiki)
- Spring官方文档
- Spring 3.0文档

# Spring hello world实例 - Spring教程

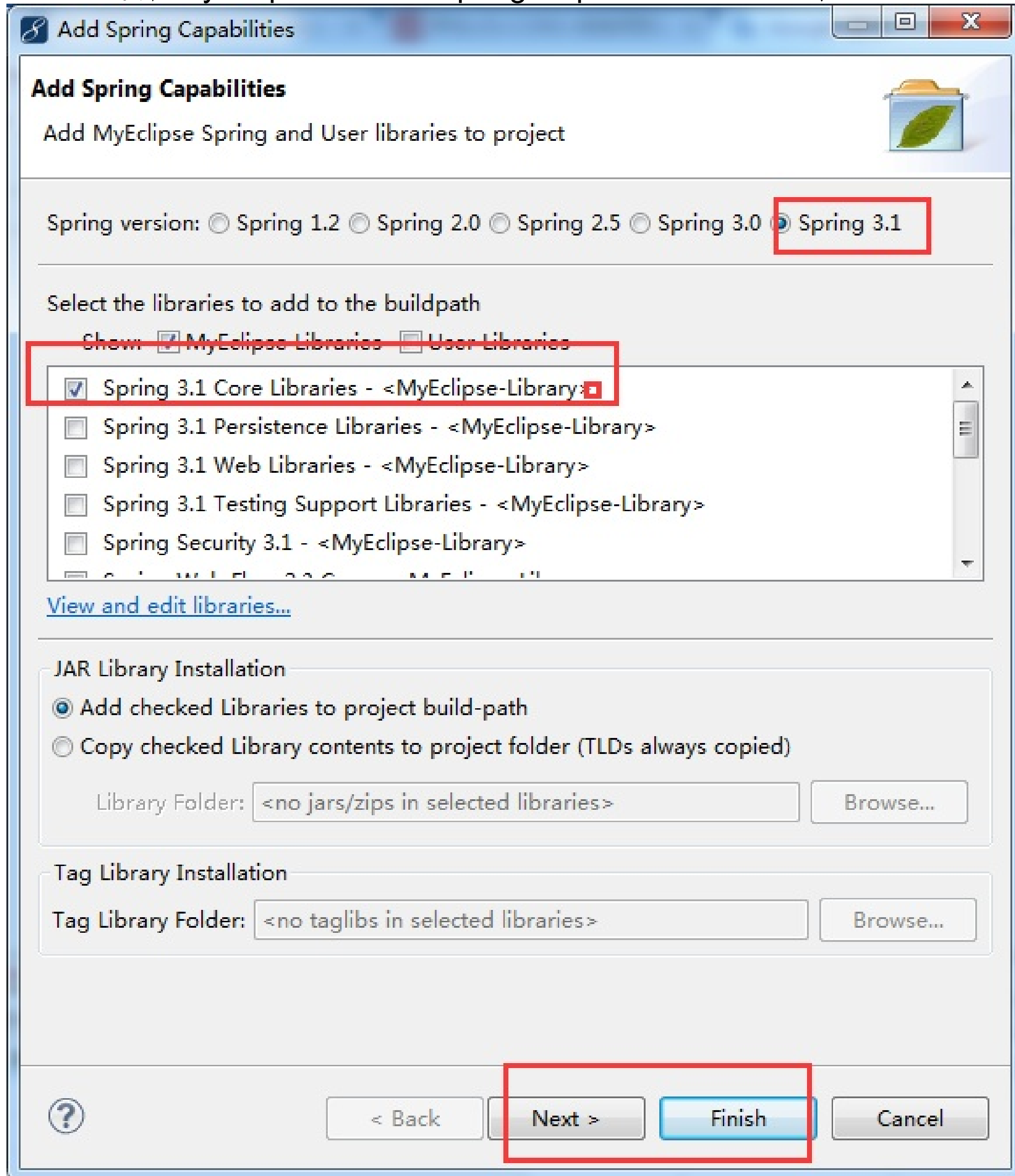
本教程介绍如何在Spring3 中创建一个简单的 Hello World 例子。在这篇文章中使用的技术或工具：

1. Spring 3.1
2. MyEclipse 10
3. JDK 1.6

提示: Spring 3 至少JDK1.5才能正常工作。Spring 3.0 dependencies 在Spring2.5.x 中，几乎整个Spring模块分组在一个单独的 spring.jar 文件中。由于Spring3中每模块被分成一个单独的 jar 文件，例如，spring-core, spring-expression, spring-context, spring-aop等

## 1. 创建一个Java工程

打开 MyEclipse 创建一个java工程：Helloword，并添加Spring支持类库，右键工程名称，选择"MyEclipse"->"Add Spring Capabilites"，如下图：




# 2. Spring bean

一个简单的 Spring bean.

```
package com.yiibai.core;

/**
 * Spring bean
 *
 */
public class HelloWorld {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public void printHello() {
        System.out.println("Spring 3 : Hello ! " + name);
    }
}
```

## 4. Spring bean 配置文件

创建Spring配置文件，并声明所有可用的Spring bean。

*File : _applicationContext*.xml_

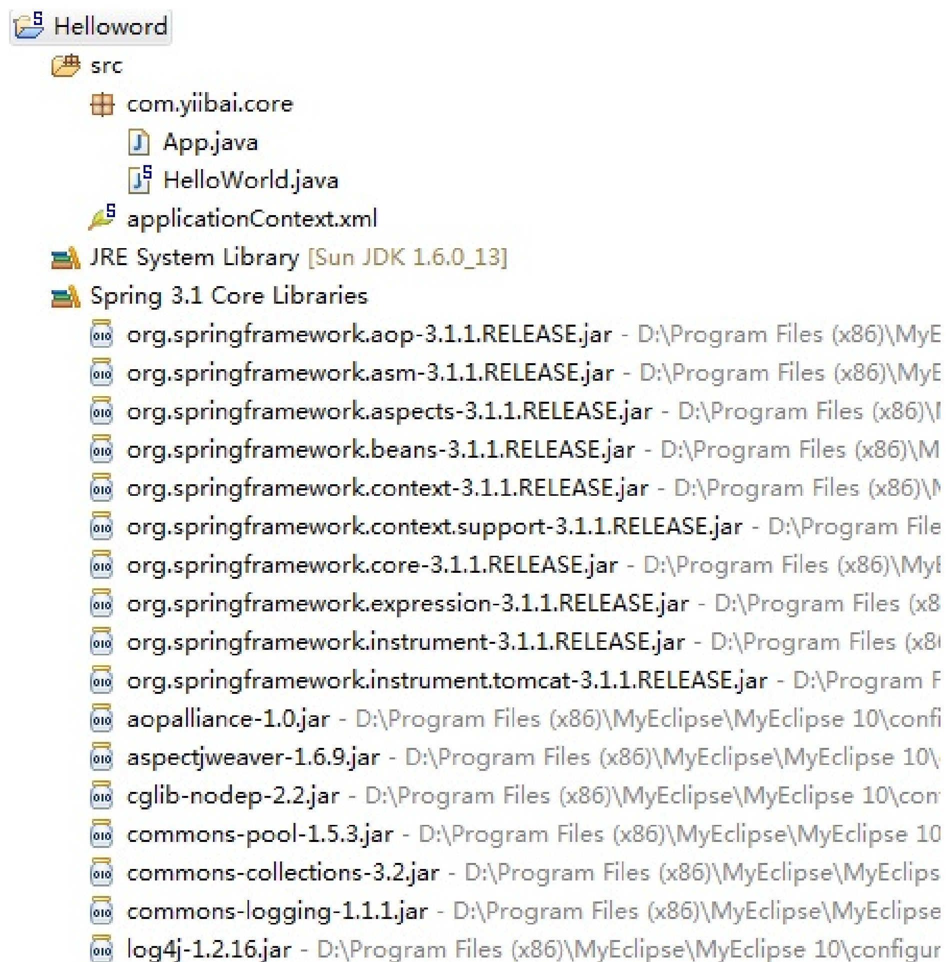
```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="helloBean" class="com.yiibai.core.HelloWorld">
        <property name="name" value="Yiibai" />
    </bean>

</beans>
```

## 5. 项目结构

查看目录结构如下:




## 6. 执行代码

```
package com.yiibai.core;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationCo
                "applicationContext.xml"); HelloWorld obj = (Hello
        obj.printHello();
    }
}
```

## 7. 输出结果

```
Spring 3 : Hello ! Yiibai
```

# 下载源代码

http://pan.baidu.com/s/1qWZQoPm

# Spring松耦合实例 - Spring教程

面向对象的概念，是一个很好的设计来打破系统进入一个组可重用的对象。然而，当系统变大，尤其是在Java项目，庞大的对象依赖关系将一直紧密耦合引起对象难以管理或修改。在这种情况下，可以使用Spring框架作为一个核心模块轻松高效地管理所有的对象依赖。

## 输出生成器的例子

让我们来看一个例子，假设你的项目有一个函数输出的内容，以CSV或JSON格式。您的代码可能看起来像下面的例子：

File : IOutputGenerator.java – 输出生成器接口

```
package com.yiibai.output;

public interface IOutputGenerator
{
    public void generateOutput();
}
```

File : CsvOutputGenerator.java – 一个CSV输出生成器用来实现IOutputGenerator接口。

```
package com.yiibai.output.impl;

import com.yiibai.output.IOutputGenerator;

public class CsvOutputGenerator implements IOutputGenerator
{
    public void generateOutput(){
        System.out.println("Csv Output Generator");
    }
}
```

File : JsonOutputGenerator.java – 一个JSON输出生成器用来实现IOutputGenerator接口。

```
package com.yiibai.output.impl;

import com.yiibai.output.IOutputGenerator;

public class JsonOutputGenerator implements IOutputGenerator
{
    public void generateOutput(){
        System.out.println("Json Output Generator");
    }
}
```

有几个方法来调用IOutputGenerator，以及如何使用 Spring 来避免对象相互结合紧密。

# 1. 方法1 – 直接调用

正常的方式，直接调用它。

```
package com.yiibai.common;

import com.yiibai.output.IOutputGenerator;
import com.yiibai.output.impl.CsvOutputGenerator;

public class App
{
    public static void main( String[] args )
    {
        IOutputGenerator output = new CsvOutputGenerator();
        output.generateOutput();
    }
}
```

存在问题

以这种方式，这个问题是“output”紧密到CsvOutputGenerator耦合，输出生成的每一个变化可能涉及代码变化。如果此代码分散在你的项目中，输出生成的每一次变化都会让你受苦。

# 方法2 – 用辅助类调用它

可能想创建一个辅助类将所有输出实现在类的内部。

```
package com.yiibai.output;

import com.yiibai.output.IOutputGenerator;
import com.yiibai.output.impl.CsvOutputGenerator;

public class OutputHelper
{
    IOutputGenerator outputGenerator;

    public OutputHelper(){
        outputGenerator = new CsvOutputGenerator();
    }

    public void generateOutput(){
        outputGenerator.generateOutput();
    }

}
```

通过辅助类调用它

```
package com.yiibai.common;

import com.yiibai.output.OutputHelper;

public class App
{
    public static void main( String[] args )
    {
        OutputHelper output = new OutputHelper();
        output.generateOutput();
    }
}
```

存在问题

这看起来比之前的更优雅，只需要管理一个辅助类，但是辅助类仍是紧耦合 CsvOutputGenerator，输出生成的每一个变化仍然涉及小的代码更改。

# 方法3 – Spring

在这种情况下，Spring 依赖注入(DI)是一个不错的选择。Spring 可以让输出生成松散的耦合到输出发生器。OutputHelper类更小的修改。

```
package com.yiibai.output;

import com.yiibai.output.IOutputGenerator;

public class OutputHelper
{
    IOutputGenerator outputGenerator;

    public void generateOutput(){
        outputGenerator.generateOutput();
    }

    public void setOutputGenerator(IOutputGenerator outputGenerato
        this.outputGenerator = outputGenerator;
    }
}
```

创建一个 Spring bean 的配置文件，并在这里声明所有的Java对象的依赖。

```
<!-- Spring-Common.xml -->
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="OutputHelper" class="com.yiibai.output.OutputHelper">
        <property name="outputGenerator" ref="CsvOutputGenerator" /
    </bean>

    <bean id="CsvOutputGenerator" class="com.yiibai.output.impl.Csv
    <bean id="JsonOutputGenerator" class="com.yiibai.output.impl.Js

</beans>
```

通过Spring来调用它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.output.OutputHelper;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
           new ClassPathXmlApplicationContext(new String[] {"Spring

        OutputHelper output = (OutputHelper)context.getBean("Output
        output.generateOutput();

    }
}
```

现在，只需要改变 Spring XML 文件使用不同的输出生成器。只修改 Spring XML 文件而不需要无码修改，这意味着更少的错误。

## 结论

有了Spring框架 - 这种依赖注入(DI)为对象的依赖关系管理有用的特性，使大型Java项目开发管理中更优雅的，高度灵活和便于维护。

# Spring JavaConfig实例 - Spring教程

从Spring 3起，JavaConfig功能已经包含在Spring核心模块，它允许开发者将bean定义和在Spring配置XML文件到Java类中。但是，仍然允许使用经典的XML方式来定义bean和配置，JavaConfig是另一种替代解决方案。看来看经典的XML定义和JavaConfig的不同，如下定义在Spring容器中的bean。

Spring XML file - applicationContext.xml :

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="helloBean" class="com.yiibai.hello.impl.HelloWorldImp

</beans>
```

等效于以下JavaConfig的配置：

```
package com.yiibai.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.yiibai.hello.HelloWorld;
import com.yiibai.hello.impl.HelloWorldImpl;

@Configuration
public class AppConfig {

    @Bean(name="helloBean")
    public HelloWorld helloWorld() {
        return new HelloWorldImpl();
    }

}
```

## Spring JavaConfig Hello World

现在，看到一个完整的Spring JavaConfig例子。

## 1. 工程目录结构

这个例子的目录结构如下。

# 3. Spring Bean

一个简单的Bean

```
package com.yiibai.hello;

public interface HelloWorld {

    void printHelloWorld(String msg);

}
```

```
package com.yiibai.hello.impl;

import com.yiibai.hello.HelloWorld;

public class HelloWorldImpl implements HelloWorld {

    @Override
    public void printHelloWorld(String msg) {

        System.out.println("Hello : " + msg);
    }

}
```

# 4. JavaConfig 注解

使用 @Configuration 注释告诉 Spring，这是核心的 Spring 配置文件，并通过 @Bean 定义 bean。

```
package com.yiibai.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.yiibai.hello.HelloWorld;
import com.yiibai.hello.impl.HelloWorldImpl;

@Configuration
public class AppConfig {

    @Bean(name="helloBean")
    public HelloWorld helloWorld() {
        return new HelloWorldImpl();
    }

}
```

## 5. 执行结果

使用 AnnotationConfigApplicationContext 加载您的JavaConfig类

```
package com.yiibai.core;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigAppli
import com.yiibai.config.AppConfig;
import com.yiibai.hello.HelloWorld;

public class App {
    public static void main(String[] args) {

            ApplicationContext context = new AnnotationConfigApplic
        HelloWorld obj = (HelloWorld) context.getBean("helloBean")

        obj.printHelloWorld("Spring Java Config");

    }
}
```

输出结果

```
Hello : Spring Java Config
```

下载代码 – http://pan.baidu.com/s/1kTToPSF

# Spring JavaConfig @Import实例 - Spring教程

一般来说，需要按模块或类别 分割Spring XML bean文件 成多个小文件，使事情更容易维护和模块化。 例如，

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <import resource="config/customer.xml"/>
        <import resource="config/scheduler.xml"/>

</beans>
```

Spring3 JavaConfig它等效于 @Import 功能

```
package com.yiibai.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

@Configuration
@Import({ CustomerConfig.class, SchedulerConfig.class })
public class AppConfig {

}
```

## @Import实例

请参阅使用JavaConfig @Import的一个完整的例子。

## 1. 目录结构

本实例目录结构。

## 2. Spring Beans

两个简单的Spring bean。

*File : CustomerBo.java*

```
package com.yiibai.core;

public class CustomerBo {

    public void printMsg(String msg) {

        System.out.println("CustomerBo : " + msg);
    }

}
```

*File : SchedulerBo.java*

```
package com.yiibai.core;

public class SchedulerBo {

    public void printMsg(String msg) {

        System.out.println("SchedulerBo : " + msg);
    }

}
```

## 3. @Configuration示例

现在，使用JavaConfig @Configuration声明上述Bean类。

*File : CustomerConfig.java*

```
package com.yiibai.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.yiibai.core.CustomerBo;

@Configuration
public class CustomerConfig {

    @Bean(name="customer")
    public CustomerBo customerBo(){

        return new CustomerBo();

    }
}
```

*File : SchedulerConfig.java*

```
package com.yiibai.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.yiibai.core.SchedulerBo;

@Configuration
public class SchedulerConfig {

    @Bean(name="scheduler")
    public SchedulerBo suchedulerBo(){

        return new SchedulerBo();

    }

}
```

## 4. @Import示例

使用@Import加载多个配置文件。

*File : AppConfig.java*

```
package com.yiibai.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

@Configuration
@Import({ CustomerConfig.class, SchedulerConfig.class })
public class AppConfig {

}
```

## 5. 执行程序

加载主配置文件，并进行测试。

```
package com.yiibai.core;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigAppli
import com.yiibai.config.AppConfig;

public class App {
    public static void main(String[] args) {

        ApplicationContext context = new AnnotationConfigApplicatio
                AppConfig.class);

        CustomerBo customer = (CustomerBo) context.getBean("custome
        customer.printMsg("Hello 11");

        SchedulerBo scheduler = (SchedulerBo) context.getBean("sche
        scheduler.printMsg("Hello 22");

    }
}
```

输出结果

```
CustomerBo : Hello 11
SchedulerBo : Hello 22
```

下载代码 – http://pan.baidu.com/s/1basJTk

# Spring依赖注入（DI） - Spring教程

在Spring框架中，依赖注入(DI)的设计模式是用来定义对象彼此间的依赖。它主要有两种类型：

- Setter方法注入
- 构造器注入

## 1. Setter方法注入

这是最流行最简单的DI注入方法，通过设置方法注入依赖。

### 示例

帮助器类和指定的setter方法

```
package com.yiibai.output;

import com.yiibai.output.IOutputGenerator;

public class OutputHelper
{
    IOutputGenerator outputGenerator;

    public void setOutputGenerator(IOutputGenerator outputGenerator
        this.outputGenerator = outputGenerator;
    }

}
```

一个 bean 配置文件用来声明bean 和通过 setter 设置注入(property标签)的依赖。

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="OutputHelper" class="com.yiibai.output.OutputHelper">
        <property name="outputGenerator">
            <ref bean="CsvOutputGenerator" />
        </property>
    </bean>

<bean id="CsvOutputGenerator" class="com.yiibai.output.impl.CsvOutp
<bean id="JsonOutputGenerator" class="com.yiibai.output.impl.JsonOu

</beans>
```

只需注入一个“CsvOutputGenerator” bean 到 “OutputHelper”对象，通过一个 setter 方法(setOutputGenerator)。

## 2.构造函数注入

此DI方法将通过构造函数注入依赖。

### 示例

一个辅助类的构造函数。

```
package com.yiibai.output;

import com.yiibai.output.IOutputGenerator;

public class OutputHelper
{
    IOutputGenerator outputGenerator;

        OutputHelper(IOutputGenerator outputGenerator){
        this.outputGenerator = outputGenerator;
    }
}
```

bean 配置文件来声明bean并通过构造函数(constructor-arg标签)设置注入依赖。

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="OutputHelper" class="com.yiibai.output.OutputHelper">
        <constructor-arg>
            <bean class="com.yiibai.output.impl.CsvOutputGenerator'
        </constructor-arg>
    </bean>

<bean id="CsvOutputGenerator" class="com.yiibai.output.impl.CsvOutp
<bean id="JsonOutputGenerator" class="com.yiibai.output.impl.JsonOu

</beans>
```

只需通过一个构造函数注入一个 “CsvOutputGenerator” Bean 到 “OutputHelper” 对象。

## setter方法还是构造函数注入？

Spring框架的设置有没有硬性规定，只需要使用适合你的项目需要的类型注入。然而，由于setter方法注入简单，它总是大部分使用场景的选择。

## 参考

1. http://en.wikipedia.org/wiki/Dependency_injection

# Spring使用Setter依赖注入 - Spring教程

一个简单的Spring例子来展示如何通过setter方法注入依赖项，最常用DI方法注入bean。

## 1. IOutputGenerator

接口和实现类

```
package com.yiibai.output;

public interface IOutputGenerator
{
    public void generateOutput();
}
```

```
package com.yiibai.output.impl;

import com.yiibai.output.IOutputGenerator;

public class CsvOutputGenerator implements IOutputGenerator {
    public void generateOutput() {
        System.out.println("This is Csv Output Generator");
    }
}
```

## 2. Helper类

一个辅助类，之后使用Spring 来注入 IOutputGenerator。

```
package com.yiibai.output;

import com.yiibai.output.IOutputGenerator;

public class OutputHelper {
    IOutputGenerator outputGenerator;

    public void generateOutput() {
        outputGenerator.generateOutput();
    }

    //DI via setter method
    public void setOutputGenerator(IOutputGenerator outputGenerator
        this.outputGenerator = outputGenerator;
    }
}
```

## 3. Spring配置

配置Bean在Spring配置文件，并引用Bean “CsvOutputGenerator” 到 “OutputHelper”，通过property 和 ref 标签。

在这种情况下，Spring将通过setter方法注入Bean “CsvOutputGenerator” 到“OutputHelper”类，“setOutputGenerator(IOutputGenerator outputGenerator)”.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="OutputHelper" class="com.yiibai.output.OutputHelper">
        <property name="outputGenerator" ref="CsvOutputGenerator" /
    </bean>

    <bean id="CsvOutputGenerator" class="com.yiibai.output.impl.Csv

</beans>
```

## 4. 执行结果

载入一切东西，并运行它。

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(
import com.yiibai.output.OutputHelper;

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationCor
                "applicationContext.xml"); OutputHelper output = ((
                output.generateOutput();
    }
}
```

输出结果

```
This is Csv Output Generator
```

下载源代码 – http://pan.baidu.com/s/1c1tR6ZQ

# Spring通过构造方法依赖注入 - Spring教程

使用Spring进行依赖，通过构造函数注入一个bean。

## 1. IOutputGenerator

接口和实现类

```
package com.yiibai.output;

public interface IOutputGenerator
{
    public void generateOutput();
}
```

```
package com.yiibai.output.impl;

import com.yiibai.output.IOutputGenerator;

public class JsonOutputGenerator implements IOutputGenerator
{
    public void generateOutput(){
        System.out.println("This is Json Output Generator");
    }
}
```

## 2. Helper类

一个辅助类，之后使用Spring 通过构造方法注入 IOutputGenerator。

```
package com.yiibai.output;

import com.yiibai.output.IOutputGenerator;

public class OutputHelper {
    IOutputGenerator outputGenerator;

    public void generateOutput() {
        outputGenerator.generateOutput();
    }

    //DI via constructor
    public OutputHelper(IOutputGenerator outputGenerator){
        this.outputGenerator = outputGenerator;
    }

}
```

## 3. Spring配置

请参阅下面的 Spring bean 配置，Spring 将通过构造函数注入上面的 “JsonOutputGenerator” 到 “OutputHelper” 类，“public OutputHelper(IOutputGenerator outputGenerator)“.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="OutputHelper" class="com.yiibai.output.OutputHelper">
        <constructor-arg>
            <ref bean="JsonOutputGenerator" />
        </constructor-arg>
    </bean>

    <bean id="JsonOutputGenerator" class="com.yiibai.output.impl.Js

</beans>
```

## 4.执行结果

载入一切东西，并运行它。

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(
import com.yiibai.output.OutputHelper;

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationCor
                "applicationContext.xml"); OutputHelper output = ((
                output.generateOutput();
    }
}
```

输出结果

```
This is Json Output Generator
```

构造方法类型歧义对于构造函数支持多个参数，会导致公共的构造注入类型歧义的问题，请阅读详细的解决方案。下载代码 – http://pan.baidu.com/s/1skwOPqd

# Spring Bean引用例子 - Spring教程

在Spring，bean可以“访问”对方通过bean配置文件指定相同或不同的引用。

## 1. Bean在不同的XML文件

如果是在不同XML文件中的bean，可以用一个“ref”标签，“bean”属性引用它。

```
<ref bean="someBean"/>
```

“CsvOutputGenerator”或“JsonOutputGenerator”通过属性标签使用'ref'属性， - 在这个例子中，Bean “OutputHelper” 在 'Spring-Common.xml' 声明可以访问其他Bean在'Spring-Output.xml'。

*File : Spring-Common.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="OutputHelper" class="com.yiibai.output.OutputHelper">
        <property name="outputGenerator" >
            <ref bean="CsvOutputGenerator"/>
        </property>
    </bean>

</beans>
```

*File : Spring-Output.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CsvOutputGenerator" class="com.yiibai.output.impl.Csv
    <bean id="JsonOutputGenerator" class="com.yiibai.output.impl.Js

</beans>
```

# 2. 在同一个XML文件中的Bean

如果引用在同一个XML文件中的bean，你可以用 'ref' 标签，“local”属性引用它。

```
<ref local="someBean"/>
```

在这个例子中，Bean “OutputHelper” 在 'Spring-Common.xml' 声明可以相互访问“CsvOutputGenerator”或“JsonOutputGenerator”。

*File : Spring-Common.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="OutputHelper" class="com.yiibai.output.OutputHelper">
        <property name="outputGenerator" >
            <ref local="CsvOutputGenerator"/>
        </property>
    </bean>

    <bean id="CsvOutputGenerator" class="com.yiibai.output.impl.Csv
    <bean id="JsonOutputGenerator" class="com.yiibai.output.impl.Js

</beans>
```

## 总结

其实，无论是在相同或不同的XML文件，“ref” 标签可以访问一个bean，但是，对于该项目的可读性，如果引用了相同的 XML文件中声明 bean，您应该使用“local”属性。

# 如何注入值到Spring bean属性 - Spring教程

在Spring中，有三种方式注入值到 bean 属性。

- 正常的方式
- 快捷方式
- **“p”** 模式

看到一个简单的Java类，它包含两个属性 - name 和 type。稍后将使用Spring注入值到这个 bean 属性。

```
package com.yiibai.common;

public class FileNameGenerator
{
    private String name;
    private String type;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
}
```

## 1.正常方式

在一个“value”标签注入值，并附有“property”标签结束。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="FileNameGenerator" class="com.yiibai.common.FileName(
        <property name="name">
            <value>yiibai</value>
        </property>
        <property name="type">
            <value>txt</value>
        </property>
    </bean>
</beans>
```

## 2，快捷方式

注入值“value”属性。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="FileNameGenerator" class="com.yiibai.common.FileName(
        <property name="name" value="yiibai" />
        <property name="type" value="txt" />
    </bean>

</beans>
```

## 3. “p” 模式

通过使用“p”模式作为注入值到一个属性。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="FileNameGenerator" class="com.yiibai.common.FileName(
             p:name="yiibai" p:type="txt" />

</beans>
```

记住声明 xmlns:p="http://www.springframework.org/schema/p" 在Spring XML bean配置文件。

# 总结

这些方法的使用完全是基于个人喜好，也不会影响注入bean属性的值。

# Spring bean加载多个配置文件 - Spring教程

在一个大的项目结构，Spring bean配置文件位于不同的文件夹以便于维护和模块化。例如，Spring-Common.xml在common 文件夹中，Spring-Connection.xml 在connection文件夹，Spring-ModuleA.xml在ModuleA 文件夹等等。你可以加载多个Spring bean的配置文件如下代码中：

```
ApplicationContext context =
        new ClassPathXmlApplicationContext(new String[] {"Spring-Co
              "Spring-Connection.xml","Spring-ModuleA.xml"});
```

把所有的 Spring XML 文件放入在项目类路径中。

```
project-classpath/Spring-Common.xml
    project-classpath/Spring-Connection.xml
    project-classpath/Spring-ModuleA.xml
```

## 解决方法

以上方法是缺乏组织并且很容易出错，更好的办法应组织所有的Spring bean 配置文件到一个XML文件。例如，创建一个Spring-All-Module.xml文件，并导入整个Spring bean的文件如下：File : Spring-All-Module.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <import resource="common/Spring-Common.xml"/>
        <import resource="connection/Spring-Connection.xml"/>
        <import resource="moduleA/Spring-ModuleA.xml"/>

</beans>
```

现在，可以加载一个这样的 XML 文件：

```
ApplicationContext context =
            new ClassPathXmlApplicationContext(Spring-All-Module.xn
```

将这个文件放入项目的类路径。

```
project-classpath/Spring-All-Module.xml
```

注意 在Spring3，所述替代解决方案是使用 JavaConfig @Import.

# Spring内部bean实例 - Spring教程

在Spring框架中，一个bean仅用于一个特定的属性，这是提醒其声明为一个内部bean。内部bean支持setter注入“property”和构造器注入"constructor-arg“。下面来看看一个详细的例子，演示使用 Spring 内部 bean 。

```
package com.yiibai.common;

public class Customer
{
    private Person person;

    public Customer(Person person) {
        this.person = person;
    }

    public void setPerson(Person person) {
        this.person = person;
    }

    @Override
    public String toString() {
        return "Customer [person=" + person + "]";
    }
}
```

```
package com.yiibai.common;

public class Person
{
    private String name;
    private String address;
    private int age;

    //getter and setter methods

    @Override
    public String toString() {
        return "Person [address=" + address + ",
                               age=" + age + ", name=" + name + "]'
    }
}
```

很多时候，可以使用 'ref' 属性来引用“Person” bean到“Customer” Bean，person的属性如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="person" ref="PersonBean" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address1" />
        <property name="age" value="28" />
    </bean>

</beans>
```

在一般情况下，引用这样也没有问题，但由于“yiibai” persion bean 只用于 Customer bean，这是更好地声明 “yiibai” person 作为一个内部 bean，如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="person">
            <bean class="com.yiibai.common.Person">
                <property name="name" value="yiibai" />
                <property name="address" value="address1" />
                <property name="age" value="28" />
            </bean>
        </property>
    </bean>
</beans>
```

内部 bean 也支持构造器注入如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <constructor-arg>
            <bean class="com.yiibai.common.Person">
                <property name="name" value="yiibai" />
                <property name="address" value="address1" />
                <property name="age" value="28" />
            </bean>
        </constructor-arg>
    </bean>
</beans>
```

注意： id 或 name 值在bean类是没有必要以一个内部 bean 呈现，它会简单地忽略Spring容器。

执行结果：

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
          new ClassPathXmlApplicationContext(new String[] {"Spring-

        Customer cust = (Customer)context.getBean("CustomerBean");
        System.out.println(cust);

    }
}
```

输出结果：

```
Customer [person=Person [address=address1, age=28, name=yiibai]]
```

# Spring Bean作用域实例 - Spring教程

在Spring中，bean作用域用于确定哪种类型的 bean 实例应该从Spring容器中返回给调用者。bean支持的5种范围域：

1. 单例 - 每个Spring IoC 容器返回一个bean实例
2. 原型- 当每次请求时返回一个新的bean实例
3. 请求 - 返回每个HTTP请求的一个Bean实例
4. 会话 - 返回每个HTTP会话的一个bean实例
5. 全局会话- 返回全局HTTP会话的一个bean实例

在大多数情况下，可能只处理了 Spring 的核心作用域 - 单例和原型，默认作用域是单例。注：意味着只有在一个基于web的Spring ApplicationContext情形下有效！

## 单例VS原型

这里有一个例子来说明，bean的作用域单例和原型之间的不同：

```
package com.yiibai.customer.services;

public class CustomerService
{
    String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

## 1.单例例子

如果 bean 配置文件中没有指定 bean 的范围，默认为单例。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

       <bean id="customerService"
            class="com.yiibai.customer.services.CustomerService" />

</beans>
```

执行结果：

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
         new ClassPathXmlApplicationContext(new String[] {"Spring-C

        CustomerService custA = (CustomerService)context.getBean("c
        custA.setMessage("Message by custA");
        System.out.println("Message : " + custA.getMessage());

        //retrieve it again
        CustomerService custB = (CustomerService)context.getBean("c
        System.out.println("Message : " + custB.getMessage());
    }
}
```

输出结果

```
Message : Message by custA
Message : Message by custA
```

由于 bean 的 “CustomerService' 是单例作用域，第二个通过提取”custB“将显示消息由 ”custA' 设置，即使它是由一个新的 getBean()方法来提取。在单例中，每个Spring IoC容器只有一个实例，无论多少次调用 getBean()方法获取它，它总是返回同一个实例。

## 2.原型例子

如果想有一个新的 “CustomerService”bean 实例，每次调用它的时候，需要使用原型(prototype)来代替。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

   <bean id="customerService" class="com.yiibai.customer.services.C
         scope="prototype"/>

</beans>
```

运行-执行

```
Message : Message by custA
Message : null
```

在原型作用域，必须为每个 getBean()方法中调用返回一个新的实例。

## 3. Bean作用域注释

还可以使用注释来定义 bean 的作用域。

```
package com.yiibai.customer.services;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Service;

@Service
@Scope("prototype")
public class CustomerService
{
    String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

启用自动组件扫描

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.

       <context:component-scan base-package="com.yiibai.customer" /

</beans>
```

下载代码 – http://pan.baidu.com/s/1o7jxMrg

# Spring集合 (List,Set,Map,Properties) 实例 - Spring教程

下面例子向您展示Spring如何注入值到集合类型(List, Set, Map, and Properties)。支持4个主要的集合类型：

- List – <list/>
- Set – <set/>
- Map – <map/>
- Properties – <props/>

## Spring beans

一个Customer对象，有四个集合属性。

```
package com.yiibai.common;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class Customer
{
    private List<Object> lists;
    private Set<Object> sets;
    private Map<Object, Object> maps;
    private Properties pros;

    //...
}
```

在bean配置文件中不同的代码片段用来声明集合。

## 1. List示例

```
<property name="lists">
        <list>
            <value>1</value>
            <ref bean="PersonBean" />
            <bean class="com.yiibai.common.Person">
                <property name="name" value="yiibaiList" />
                <property name="address" value="Hainan" />
                <property name="age" value="28" />
            </bean>
        </list>
    </property>
```

## 2. Set示例

```
<property name="sets">
        <set>
            <value>1</value>
            <ref bean="PersonBean" />
            <bean class="com.yiibai.common.Person">
                <property name="name" value="yiibaiSet" />
                <property name="address" value="Hainan" />
                <property name="age" value="28" />
            </bean>
        </set>
    </property>
```

## 3. Map示例

```
<property name="maps">
        <map>
            <entry key="Key 1" value="1" />
            <entry key="Key 2" value-ref="PersonBean" />
            <entry key="Key 3">
                <bean class="com.yiibai.common.Person">
                    <property name="name" value="yiibaiMap" />
                    <property name="address" value="Hainan" />
                    <property name="age" value="28" />
                </bean>
            </entry>
        </map>
    </property>
```

## 4. Properties示例

```
<property name="pros">
        <props>
            <prop key="admin">admin@yiibai.com</prop>
            <prop key="support">support@yiibai.com</prop>
        </props>
    </property>
```

Spring完整的 bean 配置文件。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer">

        <!-- java.util.List -->
        <property name="lists">
            <list>
                <value>1</value>
                <ref bean="PersonBean" />
                <bean class="com.yiibai.common.Person">
                    <property name="name" value="yiibaiList" />
                    <property name="address" value="Hainan Haikou"
                    <property name="age" value="28" />
                </bean>
            </list>
        </property>

        <!-- java.util.Set -->
        <property name="sets">
            <set>
                <value>1</value>
                <ref bean="PersonBean" />
                <bean class="com.yiibai.common.Person">
                    <property name="name" value="yiibaiSet" />
                    <property name="address" value="Hainan Haikou"
                    <property name="age" value="28" />
                </bean>
            </set>
        </property>

        <!-- java.util.Map -->
        <property name="maps">
            <map>
                <entry key="Key 1" value="1" />
                <entry key="Key 2" value-ref="PersonBean" />
                <entry key="Key 3">
                    <bean class="com.yiibai.common.Person">
                        <property name="name" value="yiibaiMap" />
                        <property name="address" value="Hainan Hail
```

```
                        <property name="age" value="28" />
                    </bean>
                </entry>
            </map>
        </property>

        <!-- java.util.Properties -->
        <property name="pros">
            <props>
                <prop key="admin">admin@yiibai.com</prop>
                <prop key="support">support@yiibai.com</prop>
            </props>
        </property>

    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai1" />
        <property name="address" value="Hainan Haikou 1" />
        <property name="age" value="28" />
    </bean>

</beans>
```

执行程序

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new ClassPathXmlApplicationCor

        Customer cust = (Customer)context.getBean("CustomerBean");
        System.out.println(cust);

    }
}
```

输出

```
Customer [lists=[1, [com.yiibai.common.Person@4e4ee70b](mailto:com
```

下载代码 – http://pan.baidu.com/s/1c0T3i5i

# Spring ListFactoryBean实例 - Spring教程

ListFactoryBean"类为开发者提供了一种在Spring的bean配置文件中创建一个具体的列表集合类(ArrayList和LinkedList)。这里有一个 ListFactoryBean 示例，在运行时它将实例化一个ArrayList，并注入到一个 bean 属性。

```
package com.yiibai.common;

import java.util.List;

public class Customer
{
    private List lists;
    //...
}
```

Spring bean配置文件 - applicationContext.html 文件的内容。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="lists">
            <bean class="org.springframework.beans.factory.config.L
                <property name="targetListClass">
                    <value>java.util.ArrayList</value>
                </property>
                <property name="sourceList">
                    <list>
                        <value>one</value>
                        <value>2</value>
                        <value>three</value>
                    </list>
                </property>
            </bean>
        </property>
    </bean>

</beans>
```

另外，还可以使用 util 模式和 来达到同样的目的。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-2.5.xsd"

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="lists">
            <util:list list-class="java.util.ArrayList">
                <value>one</value>
                <value>2</value>
                <value>three</value>
            </util:list>
        </property>
    </bean>

</beans>
```

请记住要包函 util 模式，否则会出现下面的错误

```
Caused by: org.xml.sax.SAXParseException:
    The prefix "util" for element "util:list" is not bound.
```

执行，查看结果：

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationCor
                "applicationContext.xml");

        Customer cust = (Customer) context.getBean("CustomerBean");
        System.out.println(cust);

    }
}
```

输出结果

```
Customer [lists=[one, 2, three]] Type=[class java.util.ArrayList]
```

在运行时实例化ArrayList并注入列表到客户的属性。下载代码 –
http://pan.baidu.com/s/1i4aK26h

# Spring SetFactoryBean实例 - Spring教程

SetFactoryBean 类为开发者提供了一种可在 Spring bean 配置文件创建一个具体的Set集合(HashSet 和 TreeSet)。这里有一个 ListFactoryBean。例如，在运行时它将实例化 HashSet，并注入到一个 bean 属性中。

```
package com.yiibai.common;

import java.util.Set;

public class Customer
{
    private Set sets;
    //...
}
```

Spring的bean配置文件。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="sets">
            <bean class="org.springframework.beans.factory.config.S
                <property name="targetSetClass">
                    <value>java.util.HashSet</value>
                </property>
                <property name="sourceSet">
                    <list>
                        <value>one</value>
                        <value>2</value>
                        <value>three</value>
                    </list>
                </property>
            </bean>
        </property>
    </bean>

</beans>
```

另外，还可以使用 util的模式 和 来做到同样的事情。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-2.5.xsd'

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="sets">
            <util:set set-class="java.util.HashSet">
                <value>one</value>
                <value>2</value>
                <value>three</value>
            </util:set>
        </property>
    </bean>

</beans>
```

请记住必须包函 util 模式，否则会出现下面的错误：

```
Caused by: org.xml.sax.SAXParseException:
    The prefix "util" for element "util:set" is not bound.
```

执行结果

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new ClassPathXmlApplicationCor

        Customer cust = (Customer)context.getBean("CustomerBean");
        System.out.println(cust);

    }
}
```

输出结果：

```
Customer [sets=[2, one, three]] Type=[class java.util.HashSet]
```

实例化 HashSet，在运行时注入到客户的set集合属性。下载代码 – http://pan.baidu.com/s/1o7woXQU

# Spring MapFactoryBean例子 - Spring教程

MapFactoryBean类为开发者提供了一种在Spring的bean配置文件中创建一个具体的Map集合类(HashMap和TreeMap)。这里有一个MapFactoryBean。例如，在运行时它将实例化一个HashMap，并注入到一个bean属性。

```
package com.yiibai.common;

import java.util.Map;

public class Customer
{
    private Map maps;
    //...
}
```

Spring 的 bean 配置文件。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="maps">
            <bean class="org.springframework.beans.factory.config.M
                <property name="targetMapClass">
                    <value>java.util.HashMap</value>
                </property>
                <property name="sourceMap">
                    <map>
                        <entry key="Key1" value="one" />
                        <entry key="Key2" value="two" />
                        <entry key="Key3" value="three" />
                    </map>
                </property>
            </bean>
        </property>
    </bean>

</beans>
```

另外，还可以使用 util 的模式和来做到同样的事情。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-2.5.xsd'

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="maps">
            <util:map map-class="java.util.HashMap">
                <entry key="Key1" value="1" />
                <entry key="Key2" value="2" />
                <entry key="Key3" value="3" />
            </util:map>
        </property>
    </bean>

</beans>
```

请记住包函util模式，否则会出现下面的错误

```
Caused by: org.xml.sax.SAXParseException:
    The prefix "util" for element "util:map" is not bound.
```

执行结果...

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new ClassPathXmlApplicationCor

        Customer cust = (Customer)context.getBean("CustomerBean");
        System.out.println(cust);

    }
}
```

输出结果：

```
Customer [maps={Key2=two, Key1=one, Key3=three}]
```

在运行时实例化一个HashMap和注入到客户的映射(Map)属性。下载代码 –
http://pan.baidu.com/s/1kTXsDoj

# Spring注入日期到bean属性-CustomDateEditor - Spring教程

这一个Spring例子向您展示如何为bean属性注入一个“日期”。

```
package com.yiibai.common;

import java.util.Date;

public class Customer {

    Date date;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    @Override
    public String toString() {
        return "Customer [date=" + date + "]";
    }

}
```

bean配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/bean
    http://www.springframework.org/schema/beans/spring-beans-2.5.x

    <bean id="customer" class="com.yiibai.common.Customer">
        <property name="date" value="2015-12-31" />
    </bean>

</beans>
```

执行-运行程序输出

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationCon
                "SpringBeans.xml");

        Customer cust = (Customer) context.getBean("customer");
        System.out.println(cust);

    }
}
```

可能会遇到如下错误信息：

```
Caused by: org.springframework.beans.TypeMismatchException:
    Failed to convert property value of type [java.lang.String] to
    required type [java.util.Date] for property 'date';

nested exception is java.lang.IllegalArgumentException:
    Cannot convert value of type [java.lang.String] to
    required type [java.util.Date] for property 'date':
    no matching editors or conversion strategy found
```

## 解决办法

在Spring中，可以通过两种方式注入日期：

# 1. Factory bean

声明一个dateFormat bean，在“customer” Bean，引用 “dateFormat” bean作为一个工厂bean。该工厂方法将调用SimpleDateFormat.parse()自动转换成字符串Date对象。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="dateFormat" class="java.text.SimpleDateFormat">
        <constructor-arg value="yyyy-MM-dd" />
    </bean>

    <bean id="customer" class="com.yiibai.common.Customer">
        <property name="date">
            <bean factory-bean="dateFormat" factory-method="parse">
                <constructor-arg value="2015-12-31" />
            </bean>
        </property>
    </bean>

</beans>
```

## 2. CustomDateEditor

声明一个 CustomDateEditor 类将字符串转换成 java.util.Date。

```
<bean id="dateEditor"
       class="org.springframework.beans.propertyeditors.CustomDateE

        <constructor-arg>
            <bean class="java.text.SimpleDateFormat">
                <constructor-arg value="yyyy-MM-dd" />
            </bean>
        </constructor-arg>
        <constructor-arg value="true" />
    </bean>
```

并声明另一个“CustomEditorConfigurer”，使 Spring转换 bean 属性，其类型为 java.util.Date。

```
<bean class="org.springframework.beans.factory.config.CustomEditor(
        <property name="customEditors">
            <map>
                <entry key="java.util.Date">
                    <ref local="dateEditor" />
                </entry>
            </map>
        </property>
    </bean>
```

bean配置文件的完整例子(applicationContext.xml)。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="dateEditor"
        class="org.springframework.beans.propertyeditors.CustomDate

        <constructor-arg>
            <bean class="java.text.SimpleDateFormat">
                <constructor-arg value="yyyy-MM-dd" />
            </bean>
        </constructor-arg>
        <constructor-arg value="true" />

    </bean>

    <bean class="org.springframework.beans.factory.config.CustomEdi
        <property name="customEditors">
            <map>
                <entry key="java.util.Date">
                    <ref local="dateEditor" />
                </entry>
            </map>
        </property>
    </bean>

    <bean id="customer" class="com.yiibai.common.Customer">
        <property name="date" value="2015-12-31" />
    </bean>

</beans>
```

下载代码 – http://pan.baidu.com/s/1bxZyfO

# Spring PropertyPlaceholderConfigurer实例 - Spring教程

很多时候，大多数Spring开发人员只是把整个部署的详细信息(数据库的详细信息，日志文件的路径)写在XML bean配置文件如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="customerDAO" class="com.yiibai.customer.dao.impl.Jdbc

        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="customerSimpleDAO" class="com.yiibai.customer.dao.imp

        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDat

        <property name="driverClassName" value="com.mysql.jdbc.Driv
        <property name="url" value="jdbc:mysql://localhost:3306/yii
        <property name="username" value="root" />
        <property name="password" value="password" />
    </bean>

</beans>
```

但是，在企业环境中，部署的细节通常只可以由系统管理员或数据库管理员来'触碰'，他们可能会拒绝直接访问你的bean的配置文件，它们会要求部署配置一个单独的文件，例如，一个简单的性能(properties)文件，仅具有部署细节。

## PropertyPlaceholderConfigurer示例

为了解决这个问题，可以使用 PropertyPlaceholderConfigurer 类通过一个特殊的格式在外部部署细节到一个属性(properties )文件，以及访问bean的配置文件 – ${variable}.

创建一个属性文件(database.properties)，包括数据库的详细信息，把它放到你的项目类路径。

```
jdbc.driverClassName=com.mysql.jdbc.Driver
    jdbc.url=jdbc:mysql://localhost:3306/yiibai_db
    jdbc.username=root
    jdbc.password=123456
```

在声明bean配置文件和提供一个PropertyPlaceholderConfigurer映射到 刚才创建的“database.properties”属性文件。

```
<bean
        class="org.springframework.beans.factory.config.PropertyPla

        <property name="location">
            <value>database.properties</value>
        </property>
    </bean>
```

完整的示例

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean
        class="org.springframework.beans.factory.config.PropertyPla

        <property name="location">
            <value>database.properties</value>
        </property>
    </bean>

    <bean id="customerDAO" class="com.yiibai.customer.dao.impl.Jdbc

        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="customerSimpleDAO"
                class="com.yiibai.customer.dao.impl.SimpleJdbcCusto

        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDat

        <property name="driverClassName" value="${jdbc.driverClassN
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

</beans>
```

可替代用法还可以使用 PropertyPlaceholderConfigurer 于某个常量，分享给所有其他bean。例如，定义在一个属性文件中的日志文件的位置，并通过 ${log.filepath} 访问不同的 bean 配置文件的属性值。

# Spring bean配置继承 - Spring教程

在 Spring，继承是用为支持bean设置一个 bean 来分享共同的值，属性或配置。一个子 bean 或继承的bean可以继承其父 bean 的配置，属性和一些属性。另外，子 Bean 允许覆盖继承的值。请参见下面的完整的例子来告诉你如何配置 bean 继承在 Spring 中工作。

```
package com.yiibai.common;

public class Customer {

    private int type;
    private String action;
    private String Country;

    //...

}
```

Bean配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="BaseCustomerMalaysia" class="com.yiibai.common.Custom
        <property name="country" value="Malaysia" />
    </bean>

    <bean id="CustomerBean" parent="BaseCustomerMalaysia">
        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

</beans>
```

以上就是“BaseCustomerMalaysia” Bean中含有的 country 属性的值，而“CustomerBean” Bean 继承其父('BaseCustomerMalaysia')这个值。

执行它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.

        Customer cust = (Customer)context.getBean("CustomerBean");
        System.out.println(cust);

    }
}
```

输出结果

```
Customer [type=1, action=buy, Country=Malaysia]
```

CustomerBean Bean 只从它的父(“BaseCustomerMalaysia”)继承 country 属性。

## 继承抽象

在上面的例子中，'BaseCustomerMalaysia' 仍然能够实例化，例如，

```
Customer cust = (Customer)context.getBean("BaseCustomerMalaysia");
```

如果你要让这个 bean 作为一个基础模板，不允许别人来实例化它，可以在一个元素中添加一个“abstract”的属性。 例如

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="BaseCustomerMalaysia" class="com.yiibai.common.Custor
        <property name="country" value="Malaysia" />
    </bean>

    <bean id="CustomerBean" parent="BaseCustomerMalaysia">
        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

</beans>
```

现在，"BaseCustomerMalaysia' Bean是一个纯粹的模板，因为Bean只能继承它，如果试图实例化它，你会遇到以下错误消息。

```
Customer cust = (Customer)context.getBean("BaseCustomerMalaysia");
```

```
org.springframework.beans.factory.BeanIsAbstractException:
    Error creating bean with name 'BaseCustomerMalaysia':
    Bean definition is abstract
```

# 纯继承模板

其实，父 bean 是不需要定义类的属性，很多时候，你可能只需要一个共同的属性共享。这里的是一个例子

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="BaseCustomerMalaysia" abstract="true">
        <property name="country" value="Malaysia" />
    </bean>

    <bean id="CustomerBean" parent="BaseCustomerMalaysia"
        class="com.yiibai.common.Customer">

        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

</beans>
```

在这种情况下，“BaseCustomerMalaysia' Bean 是一个纯粹的模板，只分享其”country“属性。

# 覆盖它

但是，仍然可以指定的子bean的新值覆盖继承的值。让我们来看看这个例子

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="BaseCustomerMalaysia" class="com.yiibai.common.Custor
        <property name="country" value="Malaysia" />
    </bean>

    <bean id="CustomerBean" parent="BaseCustomerMalaysia">
        <property name="country" value="Japan" />
        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

</beans>
```

在“CustomerBean” Bean只是覆盖父(“BaseCustomerMalaysia”)country 属性，从 ‘Malaysia’ 修改为 ‘Japan’.

```
Customer [Country=Japan, action=buy, type=1]
```

## 总结

Spring bean配置继承是为了避免多个Bean有重复共同的值或配置是非常有用的。
下载代码 – http://pan.baidu.com/s/1bIOgAq

# Spring依赖检查 - Spring教程

在Spring中，可以使用依赖检查功能，以确保所要求的属性可设置或者注入。

## 依赖检查模式

4个依赖检查支持的模式：

- none – 没有依赖检查，这是默认的模式。
- simple – 如果基本类型(int, long,double…)和集合类型(map, list..)的任何属性都没有设置，UnsatisfiedDependencyException将被抛出。
- objects – 如果对象类型的任何属性都没有设置，UnsatisfiedDependencyException将被抛出。
- all – 如果任何类型的任何属性都没有被设置，UnsatisfiedDependencyException将被抛出。

注：默认模式是 none

## 示例

Customer和Person对象的示例。

```
package com.yiibai.common;

public class Customer
{
    private Person person;
    private int type;
    private String action;

    //getter and setter methods
}
```

```
package com.yiibai.common;

public class Person
{
    private String name;
    private String address;
    private int age;

    //getter and setter methods
}
```

# 1. none 依赖检查

Spring bean配置文件使用 “none” 依赖检查模式。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer" >
        <property name="action" value="buy" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address ABC" />
        <property name="age" value="29" />
    </bean>

</beans>
```

如果没有明确定义的依赖检查模式，其默认为“none”。没有依赖检查将执行。

# 2. simple 依赖检查

Spring bean的配置文件使用“simple”依赖检查模式。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer"
         dependency-check="simple">

        <property name="person" ref="PersonBean" />
        <property name="action" value="buy" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address ABC" />
        <property name="age" value="29" />
    </bean>

</beans>
```

在“type”属性(基本类型或集合类型)，如尚未设置，UnsatisfiedDependencyException将抛出。

```
org.springframework.beans.factory.UnsatisfiedDependencyException:
Error creating bean with name 'CustomerBean'
defined in class path resource [config/Spring-Customer.xml]:
Unsatisfied dependency expressed through bean property 'type':
Set this property value or disable dependency checking for this bea
```

## 3. objects 依赖检查

Spring bean配置文件的 “objects”依赖检查模式。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer"
         dependency-check="objects">

        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address ABC" />
        <property name="age" value="29" />
    </bean>

</beans>
```

在'person'属性(对象类型)，尚未设置，一个UnsatisfiedDependencyException将抛出。

```
org.springframework.beans.factory.UnsatisfiedDependencyException:
Error creating bean with name 'CustomerBean'
defined in class path resource [config/Spring-Customer.xml]:
Unsatisfied dependency expressed through bean property 'person':
Set this property value or disable dependency checking for this bea
```

## 4. all 依赖检查

Spring bean配置文件的“all”依赖检查模式。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="CustomerBean" class="com.yiibai.common.Customer"
         dependency-check="all">

        <property name="action" value="buy" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address ABC" />
        <property name="age" value="29" />
    </bean>

</beans>
```

对“simple”和“objects”模式的组合，如果有的话类型(原型，集合和对象)的任何属性都没有设置，一个UnsatisfiedDependencyException将被抛出。

## 全局默认的依赖检查

明确定义的依赖检查模式，每个Bean配置繁琐且容易出错，可以在根元素设置一个默认的依赖性检查属性强制根元素内声明的整个bean类适用此规则。然而，这根默认模式将通过一个bean自己指定的模式下可覆盖。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    default-dependency-check="all">

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address ABC" />
        <property name="age" value="29" />
    </bean>

</beans>
```

在这个配置文件中声明所有的Bean类默都是“all”依赖检查模式。**@Required** 注解在大多数情况下，你只需要确保特定属性已经设置，但有一定是所有的类型(原始，集合或对象)属性。

# Spring使用@Required注解依赖检查 - Spring教程

Spring依赖检查 bean 配置文件用于确定的特定类型(基本，集合或对象)的所有属性被设置。在大多数情况下，你只需要确保特定属性已经设置但不是所有属性..对于这种情况，你需要 @Required 注解，请参见下面的例子：

## @Required示例

Customer对象，适用@Required在 setPerson()方法，以确保 person 属性已设置。

```
package com.yiibai.common;

import org.springframework.beans.factory.annotation.Required;

public class Customer
{
    private Person person;
    private int type;
    private String action;

    public Person getPerson() {
        return person;
    }
    @Required
    public void setPerson(Person person) {
        this.person = person;
    }
}
```

简单地套用@Required注解不会强制执行该属性的检查，还需要注册一个RequiredAnnotationBeanPostProcessor以了解在bean配置文件@Required注解。RequiredAnnotationBeanPostProcessor可以用两种方式来启用。

### 1. 包函 <context:annotation-config />

添加 Spring 上下文和 在bean配置文件。

```
<beans
    ...
    xmlns:context="http://www.springframework.org/schema/context"
    ...
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2
    ...
    <context:annotation-config />
    ...
</beans>
```

完整的实例,

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2

    <context:annotation-config />

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address ABC" />
        <property name="age" value="29" />
    </bean>

</beans>
```

## 2. 包函 RequiredAnnotationBeanPostProcessor

直接在 bean 配置文件包函“RequiredAnnotationBeanPostProcessor”。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
<bean
class="org.springframework.beans.factory.annotation.RequiredAnnotat

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

    <bean id="PersonBean" class="com.yiibai.common.Person">
        <property name="name" value="yiibai" />
        <property name="address" value="address ABC" />
        <property name="age" value="29" />
    </bean>
</beans>
```

如果你运行它，下面的错误信息会丢的，因为 person 的属性未设置。

```
org.springframework.beans.factory.BeanInitializationException:
    Property 'person' is required for bean 'CustomerBean'
```

## 结论

尝试@Required注解，它比依赖检查XML文件中更加灵活，因为它可以适用于只有一个特定属性。定义@Required 请阅读本文有关如何创建新的自定义 @Required-style 注解。

# Spring自定义@Required-style注解 - Spring教程

@Required注解是用来确保特定属性已设置。如果您迁移现有项目到Spring框架或有自己的@Required-style注解不管是什么原因，Spring允许您定义自定义@Required-style注解，相当于@Required注解。在这个例子中，您将创建一个名为 @Mandatory 定制 @Required-style 注解，相当于@Required注解。

## 1.创建@Mandatory接口

```
package com.yiibai.common;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Mandatory {
}
```

## 2.应用它到属性

```
package com.yiibai.common;

public class Customer
{
    private Person person;
    private int type;
    private String action;

    @Mandatory
    public void setPerson(Person person) {
        this.person = person;
    }
    //getter and setter methods
}
```

## 3.注册它

包函新@Mandatory注释到“RequiredAnnotationBeanPostProcessor' 类。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

<bean
class="org.springframework.beans.factory.annotation.RequiredAnnotat
    <property name="requiredAnnotationType" value="com.yiibai.commo
</bean>

    <bean id="CustomerBean" class="com.yiibai.common.Customer">
        <property name="action" value="buy" />
        <property name="type" value="1" />
    </bean>

</beans>
```

## 4. 完成

这样做，创建了一个新的自定义命名 @Required-style的@Mandatory 注解，相当于 @Required 注解。

# Spring Bean InitializingBean和DisposableBean实例 - Spring教程

在Spring中，InitializingBean和DisposableBean是两个标记接口，为Spring执行时bean的初始化和销毁某些行为时的有用方法。

1. 对于Bean实现 InitializingBean，它将运行 afterPropertiesSet()在所有的 bean 属性被设置之后。
2. 对于 Bean 实现了DisposableBean，它将运行 destroy()在 Spring 容器释放该 bean 之后。

## 示例

下面是一个例子，向您展示如何使用 InitializingBean 和 DisposableBean。一个 CustomerService bean来实现 InitializingBean和DisposableBean 接口，并有一个消息(message)属性。

```
package com.yiibai.customer.services;

import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;

public class CustomerService implements InitializingBean, Disposabl
{
    String message;

    public String getMessage() {
      return message;
    }

    public void setMessage(String message) {
      this.message = message;
    }

    public void afterPropertiesSet() throws Exception {
      System.out.println("Init method after properties are set : "
    }

    public void destroy() throws Exception {
      System.out.println("Spring Container is destroy! Customer cle
    }

}
```

*File : applicationContext.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

       <bean id="customerService" class="com.yiibai.customer.servic
        <property name="message" value="I'm property message" />
       </bean>

</beans>
```

执行它

```
package com.yiibai.common;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

import com.yiibai.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ConfigurableApplicationContext context =
            new ClassPathXmlApplicationContext(new String[] {"appli

        CustomerService cust = (CustomerService)context.getBean("cu

        System.out.println(cust);

        context.close();
    }
}
```

该ConfigurableApplicationContext.close()将关闭该应用程序的上下文，释放所有资源，并销毁所有缓存的单例bean。它是只用于 destroy() 方的演示目的。

输出结果

```
Init method after properties are set : I'm property message
com.yiibai.customer.services.CustomerService@4090c06f
Spring Container is destroy! Customer clean up
```

afterPropertiesSet()方法被调用在 message 属性设置后，而 destroy()方法是在调用 context.close()之后;建议： 不建议使用InitializingBean和DisposableBean的接口，因为它将你的代码紧耦合到 Spring 代码中。 一个更好的做法应该是在bean的配置文件属性指定 init-method和destroy-method。下载代码 – http://pan.baidu.com/s/1nu3cEXN

# Spring Bean init-method 和 destroy-method实例 - Spring教程

在Spring中，可以使用 init-method 和 destroy-method 在bean 配置文件属性用于在bean初始化和销毁某些动作时。这是用来替代 InitializingBean和DisposableBean接口。

## 示例

这里有一个例子向您展示如何使用 init-method 和 destroy-method。

```
package com.yiibai.customer.services;

public class CustomerService
{
	String message;

	public String getMessage() {
	  return message;
	}

	public void setMessage(String message) {
	  this.message = message;
	}

	public void initIt() throws Exception {
	  System.out.println("Init method after properties are set : "
	}

	public void cleanUp() throws Exception {
	  System.out.println("Spring Container is destroy! Customer cle
	}

}
```

*File : applicationContext.xml*, 在bean中定义了init-method和destroy-method属性。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="customerService" class="com.yiibai.customer.services.
        init-method="initIt" destroy-method="cleanUp">

        <property name="message" value="i'm property message" />
    </bean>

</beans>
```

执行下面的程序代码：

```
package com.yiibai.common;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

import com.yiibai.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ConfigurableApplicationContext context =
        new ClassPathXmlApplicationContext(new String[] {"applicati

        CustomerService cust = (CustomerService)context.getBean("cu

        System.out.println(cust);

        context.close();
    }
}
```

ConfigurableApplicationContext.close将关闭应用程序上下文，释放所有资源，并销毁所有缓存的单例bean。

输出

```
Init method after properties are set : I'm property message
com.yiibai.customer.services.CustomerService@5f49d886
Spring Container is destroy! Customer clean up
```

initIt()方法被调用，消息属性设置后，在 context.close()调用后，执行 cleanUp()方法；建议使用init-method 和 destroy-methodbean 在Bena配置文件，而不是执行 InitializingBean 和 DisposableBean 接口，也会造成不必要的耦合代码在Spring。
下载源代码 – http://pan.baidu.com/s/1hreksq4

# Spring @PostConstruct和@PreDestroy实例 - Spring教程

在Spring中，既可以实现 InitializingBean和DisposableBean接口或在bean配置文件中指定 init-method 和 destroy-method 在初始化和销毁回调函数。在这篇文章中，我们将介绍如何使用 @PostConstruct 和 @PreDestroy 注解来做同样的事情。

### 注：@PostConstruct和@PreDestroy 标注不属于 Spring，它是在J2EE库- common-annotations.jar。

## @PostConstruct 和 @PreDestroy

一个 CustomerService Bean使用 @PostConstruct 和 @PreDestroy 注释

```
package com.yiibai.customer.services;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

public class CustomerService
{
    String message;

    public String getMessage() {
      return message;
    }

    public void setMessage(String message) {
      this.message = message;
    }

    @PostConstruct
    public void initIt() throws Exception {
      System.out.println("Init method after properties are set : "
    }

    @PreDestroy
    public void cleanUp() throws Exception {
      System.out.println("Spring Container is destroy! Customer cle
    }

}
```

默认情况下，Spring不会意识到@PostConstruct和@PreDestroy注解。要启用它，要么注册“CommonAnnotationBeanPostProcessor”，要么在bean配置文件的‘ 指定，

# 1. CommonAnnotationBeanPostProcessor

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean class="org.springframework.context.annotation.CommonAnnot

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="message" value="i'm property message" />
    </bean>

</beans>
```

# 2. <context:annotation-config />

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.

    <context:annotation-config />

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="message" value="i'm property message" />
    </bean>

</beans>
```

执行结果

```
package com.yiibai.common;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ConfigurableApplicationContext context =
          new ClassPathXmlApplicationContext(new String[] {"Spring-

        CustomerService cust = (CustomerService)context.getBean("cu

        System.out.println(cust);

        context.close();
    }
}
```

输出结果

```
Init method after properties are set : im property message
com.yiibai.customer.services.CustomerService@47393f
...
INFO: Destroying singletons in org.springframework.beans.factory.
support.DefaultListableBeanFactory@77158a:
defining beans [customerService]; root of factory hierarchy
Spring Container is destroy! Customer clean up
```

initIt()方法(@PostConstruct)被调用时，消息属性设置后 cleanUp() 方法(@PreDestroy)是在context.close()执行后被调用;下载源代码 – http://pan.baidu.com/s/1qX2W6xl

# Spring EL hello world实例 - Spring教程

Spring EL与OGNL和JSF EL相似，计算评估或在bean创建时执行。此外，所有的Spring表达式都可以通过XML或注解。在本教程中，我们将学习如何使用Spring表达式语言(SpEL)，注入字符串，整数，Bean到属性，无论是在XML和注释。

## 1. Spring Beans

两个简单Bean，后来利用 SpEL 注入值到属性，在 XML 和 注释。

```
package com.yiibai.core;

public class Customer {

    private Item item;

    private String itemName;

}
```

```
package com.yiibai.core;

public class Item {

    private String name;

    private int qty;

}
```

## 3. Spring EL以XML形式

使用 SpEL关闭的#{ SpEL expression }括号，请参阅XML bean定义文件下面的例子。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="itemBean" class="com.yiibai.core.Item">
        <property name="name" value="itemA" />
        <property name="qty" value="10" />
    </bean>

    <bean id="customerBean" class="com.yiibai.core.Customer">
        <property name="item" value="#{itemBean}" />
        <property name="itemName" value="#{itemBean.name}" />
    </bean>

</beans>
```

1. **{itemBean} – 注入“itemBean”到“customerBean”Bean的“item”属性。**

2. **{itemBean.name} – 注入“itemBean”的“name”属性到“customerBean" bean的"itemname”属性。**

## 4. Spring EL以注解形式

请参阅等效版本注释模式。注 要在注解使用使用SpEL，必须通过注解注册您的组件。如果注册bean在XML和Java类中定义@Value，该@Value将无法执行。

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("customerBean")
public class Customer {

    @Value("#{itemBean}")
    private Item item;

    @Value("#{itemBean.name}")
    private String itemName;

    //...

}
```

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("itemBean")
public class Item {

    @Value("itemA") //inject String directly
    private String name;

    @Value("10") //inject interger directly
    private int qty;

    public String getName() {
        return name;
    }

    //...
}
```

启用自动组件扫描。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3

    <context:component-scan base-package="com.yiibai.core" />

</beans>
```

在注解模式下，可以使用@Value定义Spring EL。在这种情况下，一个String和Integer值直接注入到“itemBean”，之后又注入“itemBean”到“customerBean”属性。

## 5. 执行输出

运行它，无论是使用 SpEL在XML 还是注释都显示了同样的结果：

```
package com.yiibai.core;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationCon

        Customer obj = (Customer) context.getBean("customerBean");
        System.out.println(obj);
    }
}
```

输出结果

```
Customer [item=Item [name=itemA, qty=10], itemName=itemA]
```

下载代码 – http://pan.baidu.com/s/1kTQ7fyZ

# Spring EL bean引用实例 - Spring教程

在Spring EL，可以使用点(.)符号嵌套属性参考一个bean。例如，“bean.property_name”。

```
public class Customer {

    @Value("#{addressBean.country}")
    private String country;
```

在上面的代码片段，它从“addressBean” bean注入了“country”属性到现在的“customer”类的“country”属性的值。

## Spring EL以注解的形式

请参阅下面的例子，演示如何使用使用 SpEL 引用一个bean，bean属性也它的方法。

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("customerBean")
public class Customer {

    @Value("#{addressBean}")
    private Address address;

    @Value("#{addressBean.country}")
    private String country;

    @Value("#{addressBean.getFullAddress('yiibai')}")
    private String fullAddress;

    //getter and setter methods

    @Override
    public String toString() {
        return "Customer [address=" + address + "\n, country=" + co
                + "\n, fullAddress=" + fullAddress + "]";
    }

}
```

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("addressBean")
public class Address {

    @Value("GaoDeng, QiongShang")
    private String street;

    @Value("571100")
    private int postcode;

    @Value("CN")
    private String country;

    public String getFullAddress(String prefix) {

        return prefix + " : " + street + " " + postcode + " " + cou
    }

    //getter and setter methods

    public void setCountry(String country) {
        this.country = country;
    }

    @Override
    public String toString() {
        return "Address [street=" + street + ", postcode=" + postco
                + ", country=" + country + "]";
    }

}
```

执行结果

```
Customer obj = (Customer) context.getBean("customerBean");
System.out.println(obj);
```

输出结果

```
Customer [address=Address [street=GaoDeng, QiongShang, postcode=57
, country=CN
, fullAddress=yiibai : GaoDeng, QiongShang 571100 CN]
```

# Spring EL以XML的形式

请参阅在XML文件定义bean的等效版本。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="customerBean" class="com.yiibai.core.Customer">
        <property name="address" value="#{addressBean}" />
        <property name="country" value="#{addressBean.country}" />
        <property name="fullAddress" value="#{addressBean.getFullAd
    </bean>

    <bean id="addressBean" class="com.yiibai.core.Address">
        <property name="street" value="GaoDeng, QiongShang" />
        <property name="postcode" value="571100" />
        <property name="country" value="CN" />
    </bean>

</beans>
```

下载代码 – http://pan.baidu.com/s/1pJTgLUN

# Spring EL方法调用实例 - Spring教程

Spring表达式语言(使用SpEL)允许开发人员使用表达式来执行方法和将返回值以注入的方式到属性，或叫作“使用SpEL方法调用”。

## Spring EL在注解的形式

了解如何实现Spring EL方法调用与@Value注释。

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("customerBean")
public class Customer {

    @Value("#{'yiibai'.toUpperCase()}")
    private String name;

    @Value("#{priceBean.getSpecialPrice()}")
    private double amount;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return "Customer [name=" + name + ", amount=" + amount + "]
    }

}
```

```
package com.yiibai.core;

import org.springframework.stereotype.Component;

@Component("priceBean")
public class Price {

    public double getSpecialPrice() {
        return new Double(199.09);
    }

}
```

输出

```
Customer [name=YIIBAI, amount=199.09]
```

## 一点解释

在字串文本上调用 toUpperCase()方法。

```
@Value("#{'yiibai'.toUpperCase()}")
private String name;
```

在 ‘priceBean‘ Bean上调用getSpecialPrice() 方法

```
@Value("#{priceBean.getSpecialPrice()}")
private double amount;
```

# Spring EL在XML的形式

请参阅在XML文件定义bean的等效版本。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="customerBean" class="com.yiibai.core.Customer">
        <property name="name" value="#{'yiibai'.toUpperCase()}" />
        <property name="amount" value="#{priceBean.getSpecialPrice(
    </bean>

    <bean id="priceBean" class="com.yiibai.core.Price" />

</beans>
```

输出

```
Customer [name=YIIBAI, amount=199.09]
```

下载代码 – http://pan.baidu.com/s/1mhdwodU

# Spring EL运算符实例 - Spring教程

Spring EL支持大多数标准的数学，逻辑和关系运算符。 例如，

1. 关系运算符 – 等于 (==, eq), 不等于 (!=, ne), 小于 (<, lt), 小于或等于 (<= , le), 大于 (>, gt), 和大于或等于 (>=, ge).
2. 逻辑运算符 – 且, 或, 非 (!).
3. 数学运算符 – 加法(+), 减法 (-), 乘法 (*), 除法(/), 除模(%) 和指数幂 (^).

## Spring EL以注解的形式

这个例子说明了如何在 SpEL 中使用运算符。

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("customerBean")
public class Customer {

    //Relational operators

    @Value("#{1 == 1}") //true
    private boolean testEqual;

    @Value("#{1 != 1}") //false
    private boolean testNotEqual;

    @Value("#{1 < 1}") //false
    private boolean testLessThan;

    @Value("#{1 <= 1}") //true
    private boolean testLessThanOrEqual;

    @Value("#{1 > 1}") //false
    private boolean testGreaterThan;

    @Value("#{1 >= 1}") //true
    private boolean testGreaterThanOrEqual;

    //Logical operators , numberBean.no == 999

    @Value("#{numberBean.no == 999 and numberBean.no < 900}") //fal
    private boolean testAnd;

    @Value("#{numberBean.no == 999 or numberBean.no < 900}") //true
    private boolean testOr;
```

```
    @Value("#{!(numberBean.no == 999)}") //false
    private boolean testNot;

    //Mathematical operators

    @Value("#{1 + 1}") //2.0
    private double testAdd;

    @Value("#{'1' + '@' + '1'}") //1@1
    private String testAddString;

    @Value("#{1 - 1}") //0.0
    private double testSubtraction;

    @Value("#{1 * 1}") //1.0
    private double testMultiplication;

    @Value("#{10 / 2}") //5.0
    private double testDivision;

    @Value("#{10 % 10}") //0.0
    private double testModulus ;

    @Value("#{2 ^ 2}") //4.0
    private double testExponentialPower;

    @Override
    public String toString() {
        return "Customer [testEqual=" + testEqual + ", testNotEqual
                + testNotEqual + ", testLessThan=" + testLessThan
                + ", testLessThanOrEqual=" + testLessThanOrEqual
                + ", testGreaterThan=" + testGreaterThan
                + ", testGreaterThanOrEqual=" + testGreaterThanOrEc
                + ", testAnd=" + testAnd + ", testOr=" + testOr + '
                + testNot + ", testAdd=" + testAdd + ", testAddStr
                + testAddString + ", testSubtraction=" + testSubtra
                + ", testMultiplication=" + testMultiplication
                + ", testDivision=" + testDivision + ", testModulus
                + testModulus + ", testExponentialPower="
                + testExponentialPower + "]";
    }

}
```

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("numberBean")
public class Number {

    @Value("999")
    private int no;

    public int getNo() {
        return no;
    }

    public void setNo(int no) {
        this.no = no;
    }

}
```

执行

```
Customer obj = (Customer) context.getBean("customerBean");
System.out.println(obj);
```

输出结果

```
Customer [testEqual=true, testNotEqual=false, testLessThan=false, t
```

## Spring EL以XML形式

请参阅在XML文件定义bean的等效版本。在XML中，类似 < 小于符号不支持，应该使用下面所示文本形式，例如文本等值，('<' = 'lt') 和 ('<=' = 'le').

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="customerBean" class="com.yiibai.core.Customer">

      <property name="testEqual" value="#{1 == 1}" />
      <property name="testNotEqual" value="#{1 != 1}" />
      <property name="testLessThan" value="#{1 lt 1}" />
      <property name="testLessThanOrEqual" value="#{1 le 1}" />
      <property name="testGreaterThan" value="#{1 > 1}" />
      <property name="testGreaterThanOrEqual" value="#{1 >= 1}" />

      <property name="testAnd" value="#{numberBean.no == 999 and nu
      <property name="testOr" value="#{numberBean.no == 999 or numb
      <property name="testNot" value="#{!(numberBean.no == 999)}" /

      <property name="testAdd" value="#{1 + 1}" />
      <property name="testAddString" value="#{'1' + '@' + '1'}" />
      <property name="testSubtraction" value="#{1 - 1}" />
      <property name="testMultiplication" value="#{1 * 1}" />
      <property name="testDivision" value="#{10 / 2}" />
      <property name="testModulus" value="#{10 % 10}" />
      <property name="testExponentialPower" value="#{2 ^ 2}" />

    </bean>

    <bean id="numberBean" class="com.yiibai.core.Number">
        <property name="no" value="999" />
    </bean>

</beans>
```

下载代码 - http://pan.baidu.com/s/1MLG1w

# Spring EL三元运算(if-then-else)实例 - Spring教程

Spring EL支持三元运算符，执行“if then else”条件检查。 例如，

```
condition ? true : false
```

## Spring EL以注解形式

Spring EL三元运算符可使用@Value注解。在这个例子中，如果“itemBean.qtyOnHand”小于100，则设置“customerBean.warning”为true，否则将其设置为false。

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("customerBean")
public class Customer {

    @Value("#{itemBean.qtyOnHand < 100 ? true : false}")
    private boolean warning;

    public boolean isWarning() {
        return warning;
    }

    public void setWarning(boolean warning) {
        this.warning = warning;
    }

    @Override
    public String toString() {
        return "Customer [warning=" + warning + "]";
    }

}
```

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("itemBean")
public class Item {

    @Value("99")
    private int qtyOnHand;

    public int getQtyOnHand() {
        return qtyOnHand;
    }

    public void setQtyOnHand(int qtyOnHand) {
        this.qtyOnHand = qtyOnHand;
    }

}
```

输出

```
Customer [warning=true]
```

# Spring EL以XML形式

请参阅在XML文件定义bean的等效版本。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="customerBean" class="com.yiibai.core.Customer">
        <property name="warning"
                      value="#{itemBean.qtyOnHand < 100 ? true
    </bean>

    <bean id="itemBean" class="com.yiibai.core.Item">
        <property name="qtyOnHand" value="99" />
    </bean>

</beans>
```

输出结果

```
Customer [warning=true]
```

在XML中，需要小于运算符使用"<"替换“<”。下载代码 - http://pan.baidu.com/s/1qXjtJru

# Spring EL Lists,Maps实例 - Spring教程

在这篇文章中，我们将介绍如何使用Spring EL从 Map 和 List 中获得值。事实上，使用SpEL与 Map 和 List 的工作方式与Java是完全一样的。请参阅例如：

```
//get map whete key = 'MapA'
@Value("#{testBean.map['MapA']}")
private String mapA;

//get first value from list, list is 0-based.
@Value("#{testBean.list[0]}")
private String list;
```

## Spring EL以注解的形式

在这里，创建了一个HashMap和ArrayList，并添加了一些初始测试数据。

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("customerBean")
public class Customer {

    @Value("#{testBean.map['MapA']}")
    private String mapA;

    @Value("#{testBean.list[0]}")
    private String list;

    public String getMapA() {
        return mapA;
    }

    public void setMapA(String mapA) {
        this.mapA = mapA;
    }

    public String getList() {
        return list;
    }

    public void setList(String list) {
        this.list = list;
    }

    @Override
    public String toString() {
        return "Customer [mapA=" + mapA + ", list=" + list + "]";
    }

}
```

```
package com.yiibai.core;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.stereotype.Component;

@Component("testBean")
public class Test {

    private Map<String, String> map;
    private List<String> list;

    public Test() {
        map = new HashMap<String, String>();
        map.put("MapA", "This is MapA");
        map.put("MapB", "This is MapB");
        map.put("MapC", "This is MapC");

        list = new ArrayList<String>();
        list.add("List0");
        list.add("List1");
        list.add("List2");

    }

    public Map<String, String> getMap() {
        return map;
    }

    public void setMap(Map<String, String> map) {
        this.map = map;
    }

    public List<String> getList() {
        return list;
    }

    public void setList(List<String> list) {
        this.list = list;
    }

}
```

执行程序

```
Customer obj = (Customer) context.getBean("customerBean");
System.out.println(obj);
```

输出结果:

```
Customer [mapA=This is MapA, list=List0]
```

# Spring EL以XML的形式

请参阅在XML文件定义bean的等效版本。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="customerBean" class="com.yiibai.core.Customer">
        <property name="mapA" value="#{testBean.map['MapA']}" />
        <property name="list" value="#{testBean.list[0]}" />
    </bean>

    <bean id="testBean" class="com.yiibai.core.Test" />

</beans>
```

下载代码 – http://pan.baidu.com/s/1gdTxfKv

# Spring EL正则表达式实例 - Spring教程

Spring EL支持正则表达式，可使用一个简单的关键词“matches”。如下实例，

```
@Value("#{'100' matches '\\d+' }")
private boolean isDigit;
```

它测试'100'是否是通过正则表达式‘\d+‘测试过的一个有效的数字。

## Spring EL以注解的形式

请参阅下面的 Spring EL 正则表达式的例子，这里有部分掺入三元运算符，这使得 Spring EL 非常灵活，功能强大。下面的例子应该是不言自明的。

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("customerBean")
public class Customer {

    // email regular expression
    String emailRegEx = "^[_A-Za-z0-9-]+(\\.[_A-Za-z0-9-]+)" +
            "*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";

    // if this is a digit?
    @Value("#{'100' matches '\\d+' }")
    private boolean validDigit;

    // if this is a digit + ternary operator
    @Value("#{ ('100' matches '\\d+') == true ? " +
            "'yes this is digit' : 'No this is not a digit'  }")
    private String msg;

    // if this emailBean.emailAddress contains a valid email addres
    @Value("#{emailBean.emailAddress matches customerBean.emailRegE
    private boolean validEmail;

    //getter and setter methods, and constructor
}
```

```
package com.yiibai.core;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("emailBean")
public class Email {

    @Value("admin@yiibai.com")
    String emailAddress;

    //...
}
```

输出

```
Customer [isDigit=true, msg=yes this is digit, isValidEmail=true]
```

## Spring EL以XML的形式

请参阅在XML文件定义bean的等效版本。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="customerBean" class="com.yiibai.core.Customer">
      <property name="validDigit" value="#{'100' matches '\d+' }" /
      <property name="msg"
        value="#{ ('100' matches '\d+') == true ? 'yes this is digi
      <property name="validEmail"
        value="#{emailBean.emailAddress matches '^[_A-Za-z0-9-]+(\.
    </bean>

    <bean id="emailBean" class="com.yiibai.core.Email">
      <property name="emailAddress" value="admin@yiibai.com" />
    </bean>

</beans>
```

下载代码 – http://pan.baidu.com/s/1jHkIXt0

## 参考

1. Spring EL三元运算符(if-then-else)实例

# Spring自动扫描组件 - Spring教程

通常情况下，声明所有的Bean类或组件的XML bean配置文件，这样Spring容器可以检测并注册Bean类或组件。 其实，Spring是能够自动扫描，检测和预定义的项目包并实例化bean，不再有繁琐的Bean类声明在XML文件中。下面是一个简单的Spring项目，包括客户服务和DAO层。让我们来探讨手动申明组件和自动扫描组件之间的不同。

## 1、手动声明组件

看到在 Spring 的一个正常方式来声明一个 bean。

一个正常的 bean.

```
package com.yiibai.customer.dao;

public class CustomerDAO
{
    @Override
    public String toString() {
        return "Hello , This is CustomerDAO";
    }
}
```

DAO 层.

```
package com.yiibai.customer.services;

import com.yiibai.customer.dao.CustomerDAO;

public class CustomerService
{
    CustomerDAO customerDAO;

    public void setCustomerDAO(CustomerDAO customerDAO) {
        this.customerDAO = customerDAO;
    }

    @Override
    public String toString() {
        return "CustomerService [customerDAO=" + customerDAO + "]";
    }

}
```

bean配置文件(applicationContext.xml)，在Spring中的一个普通 bean 配置。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="customerDAO" ref="customerDAO" />
    </bean>

    <bean id="customerDAO" class="com.yiibai.customer.dao.CustomerD

</beans>
```

执行程序

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
          new ClassPathXmlApplicationContext(new String[] {"Spring-

        CustomerService cust = (CustomerService)context.getBean("cu
        System.out.println(cust);

    }
}
```

输出结果

```
CustomerService [customerDAO=Hello , This is CustomerDAO]
```

## 2. 自动组件扫描

现在，启用Spring组件扫描功能。使用@Component注释来表示这是类是一个自动扫描组件。

```
package com.yiibai.customer.dao;

import org.springframework.stereotype.Component;

@Component
public class CustomerDAO
{
    @Override
    public String toString() {
        return "Hello , This is CustomerDAO";
    }
}
```

DAO层，添加@Component，表明这也是一个自动扫描组件。

```
package com.yiibai.customer.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.yiibai.customer.dao.CustomerDAO;

@Component
public class CustomerService
{
    @Autowired
    CustomerDAO customerDAO;

    @Override
    public String toString() {
        return "CustomerService [customerDAO=" + customerDAO + "]"
    }
}
```

将这个“context:component”在bean配置文件，这意味着，在 Spring 中启用自动扫描功能。base-package 是指明存储组件，Spring将扫描该文件夹，并找出Bean(注解为@Component)并注册到 Spring 容器。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.

    <context:component-scan base-package="com.yiibai.customer" />

</beans>
```

执行它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
           new ClassPathXmlApplicationContext(new String[] {"Spring

        CustomerService cust = (CustomerService)context.getBean("cu
        System.out.println(cust);

    }
}
```

输出结果

```
CustomerService [customerDAO=Hello , This is CustomerDAO]
```

这是 Spring 中的自动扫描组件如何工作。

# 自定义自动扫描组件名称

默认情况下，Spring 将小写部件的第一字符-
从'CustomerService'到'CustomerService'。可以检索该组件名称
为“CustomerService”。

```
CustomerService cust = (CustomerService)context.getBean("customerSe
```

要创建组件的自定义名称，你可以这样自定义名称：

```
@Service("AAA")
public class CustomerService
...
```

现在，可以用'AAA'这个名称进行检索。

```
CustomerService cust = (CustomerService)context.getBean("AAA");
```

# 自动组件扫描注释类型

在Spring2.5中，有4种类型的组件自动扫描注释类型

- @Component – 指示自动扫描组件。
- @Repository – 表示在持久层DAO组件。
- @Service – 表示在业务层服务组件。
- @Controller – 表示在表示层控制器组件。

因此，使用哪一个？其实并不那么重要。参见 @Repository，@Service 或
@Controller 源代码。

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Repository {

    String value() default "";

}
```

你可能会发现，所有的 @Repository, @Service 或 @Controller 被注解为
@Component。因此，我们可以只使用 @Component 对所有组件进行自动扫描？
是的，Spring会自动扫描所有组件的 @Component 注解。

它工作正常，但不是一个好的做法，为便于阅读，应该始终声明@Repository，@
Service 或 @Controller 在指定的层，使你的代码更易于阅读，如下：

*DAO 层*

```
package com.yiibai.customer.dao;

import org.springframework.stereotype.Repository;

@Repository
public class CustomerDAO
{
    @Override
    public String toString() {
        return "Hello , This is CustomerDAO";
    }
}
```

*Service 层*

```
package com.yiibai.customer.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.yiibai.customer.dao.CustomerDAO;

@Service
public class CustomerService
{
    @Autowired
    CustomerDAO customerDAO;

    @Override
    public String toString() {
        return "CustomerService [customerDAO=" + customerDAO + "]";
    }

}
```

下载代码 – http://pan.baidu.com/s/1o7hwcHW

# Spring过滤器组件自动扫描 - Spring教程

在这个Spring自动组件扫描的教程，您已经了解如何使Spring自动扫描您的组件。在这篇文章中，我们将展示如何使用组件过滤器自动扫描过程。

## 1.过滤组件 - 包含

参见下面的例子中使用Spring "过滤" 扫描并注册匹配定义"regex"，即使该类组件的名称未标注 @Component 。

DAO 层

```
package com.yiibai.customer.dao;

public class CustomerDAO
{
    @Override
    public String toString() {
        return "Hello , This is CustomerDAO";
    }
}
```

Service 层

```
package com.yiibai.customer.services;

import org.springframework.beans.factory.annotation.Autowired;
import com.yiibai.customer.dao.CustomerDAO;

public class CustomerService
{
    @Autowired
    CustomerDAO customerDAO;

    @Override
    public String toString() {
        return "CustomerService [customerDAO=" + customerDAO + "]";
    }

}
```

Spring 过滤

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.

    <context:component-scan base-package="com.yiibai" >

        <context:include-filter type="regex"
                       expression="com.yiibai.customer.dao.*DAO.*"

        <context:include-filter type="regex"
                       expression="com.yiibai.customer.services.*Se

    </context:component-scan>

</beans>
```

执行

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

import com.yiibai.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
        new ClassPathXmlApplicationContext(new String[] {"Spring-Au

        CustomerService cust = (CustomerService)context.getBean("cu
        System.out.println(cust);

    }
}
```

输出

```
CustomerService [customerDAO=Hello , This is CustomerDAO]
```

在这个XML过滤中，所有文件的名称中包含 DAO 或 Service(*DAO.*, *Services.*) 单词将被检测并在 Spring 容器中注册。

## 2.过滤组件 - 不包含

另外，您还可以排除指定组件，以避免 Spring 检测和 Spring 容器注册。不包括在这些文件中标注有 @Service 。

```
<context:component-scan base-package="com.yiibai.customer" >
        <context:exclude-filter type="annotation"
            expression="org.springframework.stereotype.Service" />
    </context:component-scan>
```

不包括那些包含DAO这个词组文件名。

```
<context:component-scan base-package="com.yiibai" >
        <context:exclude-filter type="regex"
            expression="com.yiibai.customer.dao.*DAO.*" />
    </context:component-scan>
```

下载代码 – http://pan.baidu.com/s/1pKnzB2F

# Spring自动装配Beans - Spring教程

在Spring框架，可以用 auto-wiring 功能会自动装配Bean。要启用它，只需要在 定义“autowire”属性。

```
<bean id="customer" class="com.yiibai.common.Customer" autowire="by
```

在Spring中，支持 5 自动装配模式。

- no – 缺省情况下，自动配置是通过“ref”属性手动设定
- byName – 根据属性名称自动装配。如果一个bean的名称和其他bean属性的名称是一样的，将会自装配它。
- byType – 按数据类型自动装配。如果一个bean的数据类型是用其它bean属性的数据类型，兼容并自动装配它。
- constructor – 在构造函数参数的byType方式。
- autodetect – 如果找到默认的构造函数，使用“自动装配用构造”; 否则，使用“按类型自动装配”。

## 示例

Customer 和 Person 对象自动装配示范。

```
package com.yiibai.common;

public class Customer
{
    private Person person;

    public Customer(Person person) {
        this.person = person;
    }

    public void setPerson(Person person) {
        this.person = person;
    }
    //...
}
```

```
package com.yiibai.common;

public class Person
{
    //...
}
```

## 1. Auto-Wiring ‘no’

这是默认的模式，你需要通过 'ref' 属性来连接 bean。

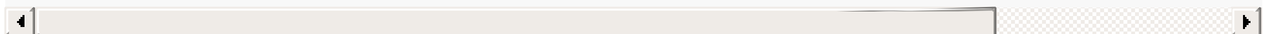
```
<bean id="customer" class="com.yiibai.common.Customer">
                  <property name="person" ref="person" />
    </bean>

    <bean id="person" class="com.yiibai.common.Person" />
```

## 2. Auto-Wiring ‘byName’

按属性名称自动装配。在这种情况下，由于对“person” bean的名称是相同于“customer” bean 的属性(“person”)名称，所以，Spring会自动通过setter方法将其装配 – “setPerson(Person person)“.

```
<bean id="customer" class="com.yiibai.common.Customer" autowire="by
<bean id="person" class="com.yiibai.common.Person" />
```

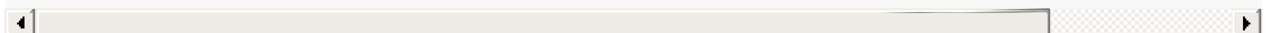


查看完整的示例 – Spring按名称自动装配

## 3. Auto-Wiring ‘byType’

通过按属性的数据类型自动装配Bean。在这种情况下，由于“Person” bean中的数据类型是与“customer” bean的属性(Person对象)的数据类型一样的，所以，Spring会自动通过setter方法将其自动装配。– “setPerson(Person person)“.

```
<bean id="customer" class="com.yiibai.common.Customer" autowire="by
<bean id="person" class="com.yiibai.common.Person" />
```



查看完整的示例 – Spring通过类型自动装配

# 4. Auto-Wiring ‘constructor’

通过构造函数参数的数据类型按属性自动装配Bean。在这种情况下，由于“person” bean的数据类型与“customer” bean的属性(Person对象)的构造函数参数的数据类型是一样的，所以，Spring通过构造方法自动装配 – “public Customer(Person person)“.

```
<bean id="customer" class="com.yiibai.common.Customer" autowire="co

    <bean id="person" class="com.yiibai.common.Person" />
```

查看完整的示例 – “. </p> <pre><bean id=)

[查看完整的示例 – ](http://www.yiibai.com/spring/spring-autowiring-by-constructor.html>Spring通过构造方法自动装配</a> ; </p>

```
    &lt;h2&gt;
        5\. Auto-Wiring ‘autodetect’
    &lt;/h2&gt;

    &lt;p&gt;
        如果找到默认的构造函数，则使用“构造方法”; 否则，使用“byType”。在这种
    &lt;/p&gt;
```

<pre><bean id=)Spring按AutoDetect自动装配成功.

注这是一件好事，这两个auto-wire’ 和 ‘dependency-check’ 相结合，以确保属性始终自动装配成功。

```
<bean id="customer" class="com.yiibai.common.Customer"
            autowire="autodetect" dependency-check="objects />

    <bean id="person" class="com.yiibai.common.Person" />
```

## 结论

在我看来，Spring ‘auto-wiring’ 以极大的成本做出更快开发效率 - 它增加了对整个 bean 配置文件复杂性，甚至不知道哪些bean将自动装配哪个Bean。

在实践中，我更顷向手动关联创建，它始终是干净，也很好地工作，或者使用 @Autowired 注解，这是更加灵活和建议。

# Spring AOP通知实例 – Advice - Spring教程

Spring AOP(面向方面编程)框架，用于在模块化方面的横切关注点。简单得说，它只是一个拦截器拦截一些过程，例如，当一个方法执行，Spring AOP 可以劫持一个执行的方法，在方法执行之前或之后添加额外的功能。在Spring AOP中，有 4 种类型通知(advices)的支持：

- 通知(Advice)之前 - 该方法执行前运行
- 通知(Advice)返回之后 – 运行后，该方法返回一个结果
- 通知(Advice)抛出之后 – 运行方法抛出异常后，
- 环绕通知 – 环绕方法执行运行，结合以上这三个通知。

下面的例子显示Spring AOP 通知如何工作。

## 简单的 Spring 例子

创建一个简单的客户服务类及一个print方法作为演示。

```
package com.yiibai.customer.services;

public class CustomerService {
    private String name;
    private String url;

    public void setName(String name) {
        this.name = name;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public void printName() {
        System.out.println("Customer name : " + this.name);
    }

    public void printURL() {
        System.out.println("Customer website : " + this.url);
    }

    public void printThrowException() {
        throw new IllegalArgumentException();
    }

}
```

File : applicationContext.xml – 一个bean配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="YiiBaii Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

</beans>
```

执行它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.customer.services.CustomerService;

public class App {
    public static void main(String[] args) {
        ApplicationContext appContext = new ClassPathXmlApplicatior
                new String[] { "Spring-Customer.xml" });

        CustomerService cust = (CustomerService) appContext.getBear

        System.out.println("*************************");
        cust.printName();
        System.out.println("*************************");
        cust.printURL();
        System.out.println("*************************");
        try {
            cust.printThrowException();
        } catch (Exception e) {

        }

    }
}
```

输出

```
*************************
Customer name : Yiibai Mook Kim
*************************
Customer website : http://www.yiibai.com
*************************
```

一个简单的Spring项目用来注入(DI)bean和输出一些字符串。

## Spring AOP 通知

现在，附加 Spring AOP 建议到上述的客户服务。

## 1. 之前通知

它会在方法执行之前执行。创建一个实现 MethodBeforeAdvice 接口的类。

```
package com.yiibai.aop;

import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;

public class HijackBeforeMethod implements MethodBeforeAdvice
{
    @Override
    public void before(Method method, Object[] args, Object target)
        throws Throwable {
            System.out.println("HijackBeforeMethod : Before method
    }
}
```

在 bean 配置文件(applicationContext.xml)，创建一个 bean 的 HijackBeforeMethod 类，并命名为“customerServiceProxy” 作为一个新的代理对象。

- ‘target’ – 定义你想拦截的bean。
- ‘interceptorNames’ – 定义要应用这个代理/目标对象的类(通知)。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yiibai Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackBeforeMethodBean" class="com.yiibai.aop.HijackE

    <bean id="customerServiceProxy"
                 class="org.springframework.aop.framework.ProxyFact

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>hijackBeforeMethodBean</value>
            </list>
        </property>
    </bean>
</beans>
```

再次运行它，现在得到新的 customerServiceProxy bean，而不是原来的 CustomerService bean。

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC
import com.yiibai.customer.services.CustomerService;

public class App {
    public static void main(String[] args) {
        ApplicationContext appContext = new ClassPathXmlApplicatior
                new String[] { "Spring-Customer.xml" });

        CustomerService cust =
                                (CustomerService) appContext.getBea

        System.out.println("*************************");
        cust.printName();
        System.out.println("*************************");
        cust.printURL();
        System.out.println("*************************");
        try {
            cust.printThrowException();
        } catch (Exception e) {

        }

    }
}
```

输出结果

```
*************************
HijackBeforeMethod : Before method hijacked!
Customer name : Yiibai Mook Kim
*************************
HijackBeforeMethod : Before method hijacked!
Customer website : http://www.yiibai.com
*************************
HijackBeforeMethod : Before method hijacked!
```

它将运行 HijackBeforeMethod 的 before() 方法，在每个 CustomerService 的方法之前执行。

## 2.返回后通知

该方法返回一个结果之后它将执行。创建一个实现AfterReturningAdvice接口的类。

```
package com.yiibai.aop;

import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;

public class HijackAfterMethod implements AfterReturningAdvice
{
    @Override
    public void afterReturning(Object returnValue, Method method,
        Object[] args, Object target) throws Throwable {
            System.out.println("HijackAfterMethod : After method hi
    }
}
```

bean配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yong Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackAfterMethodBean" class="com.yiibai.aop.HijackAf

    <bean id="customerServiceProxy"
                class="org.springframework.aop.framework.ProxyFacto

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>hijackAfterMethodBean</value>
            </list>
        </property>
    </bean>
</beans>
```

再次运行，输出

```
*************************
Customer name : Yiibai Mook Kim
HijackAfterMethod : After method hijacked!
*************************
Customer website : http://www.yiibai.com
HijackAfterMethod : After method hijacked!
*************************
```

它将运行 HijackAfterMethod 的 afterReturning()方法，在每次 CustomerService 方法返回结果之后。

# 3.抛出后通知

它将在执行方法抛出一个异常后。创建一个实现ThrowsAdvice接口的类，并创建一个afterThrowing方法拦截抛出：IllegalArgumentException异常。

```
package com.yiibai.aop;

import org.springframework.aop.ThrowsAdvice;

public class HijackThrowException implements ThrowsAdvice {
    public void afterThrowing(IllegalArgumentException e) throws Th
        System.out.println("HijackThrowException : Throw exception
    }
}
```

bean配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yong Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackThrowExceptionBean" class="com.yiibai.aop.Hijac

    <bean id="customerServiceProxy"
                class="org.springframework.aop.framework.ProxyFact

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>hijackThrowExceptionBean</value>
            </list>
        </property>
    </bean>
</beans>
```

再次运行，输出

```
*************************
Customer name : Yiibai Mook Kim
*************************
Customer website : http://www.yiibai.com
*************************
HijackThrowException : Throw exception hijacked!
```

它将运行 HijackThrowException 的 afterThrowing()方法，如果 CustomerService 的方法抛出异常。

## 4.环绕通知

它结合了上面的三个通知，在方法执行过程中执行。创建一个实现了 MethodInterceptor接口的类。必须调用“methodInvocation.proceed();” 继续在原来的方法执行，否则原来的方法将不会执行。

```
package com.yiibai.aop;

import java.util.Arrays;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class HijackAroundMethod implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation methodInvocation) throws

        System.out.println("Method name : "
                + methodInvocation.getMethod().getName());
        System.out.println("Method arguments : "
                + Arrays.toString(methodInvocation.getArguments()))

        // same with MethodBeforeAdvice
        System.out.println("HijackAroundMethod : Before method hija

        try {
            // proceed to original method call
            Object result = methodInvocation.proceed();

            // same with AfterReturningAdvice
            System.out.println("HijackAroundMethod : Before after h

            return result;

        } catch (IllegalArgumentException e) {
            // same with ThrowsAdvice
            System.out.println("HijackAroundMethod : Throw exceptio
            throw e;
        }
    }
}
```

bean配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2

    <bean id="customerService" class="com.yiibai.customer.services
        <property name="name" value="Yong Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackAroundMethodBean" class="com.yiibai.aop.Hijack

    <bean id="customerServiceProxy"
                class="org.springframework.aop.framework.ProxyFact

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>hijackAroundMethodBean</value>
            </list>
        </property>
    </bean>
</beans>
```

再次运行，输出

```
*************************
Method name : printName
Method arguments : []
HijackAroundMethod : Before method hijacked!
Customer name : YiiBai Mook Kim
HijackAroundMethod : Before after hijacked!
*************************
Method name : printURL
Method arguments : []
HijackAroundMethod : Before method hijacked!
Customer website : http://www.yiibai.com
HijackAroundMethod : Before after hijacked!
*************************
Method name : printThrowException
Method arguments : []
HijackAroundMethod : Before method hijacked!
HijackAroundMethod : Throw exception hijacked!
```

它将运行HijackAroundMethod 的 invoke()方法，在每一个 CustomerService 方法执行后。

# 总结

大部分的 Spring 开发者都只是实现了“环绕通知”，因为它可以对所有通知类型，但更好的做法应该是选择最合适的通知类型来满足要求。切入点在这个例子中，在一客户服务类中的所有方法都自动拦截(通知)。但在大多数情况下，可能需要使用切入点和Advisor通过它的方法名拦截它的方法。下载代码 –

# Spring AOP实例(Pointcut,Advisor) - Spring教程

在上一个Spring AOP通知的例子，一个类的整个方法被自动拦截。但在大多数情况下，可能只需要一种方式来拦截一个或两个方法，这就是为什么引入'切入点'的原因。它允许你通过它的方法名来拦截方法。另外，一个“切入点”必须具有“Advisor' 相关联。在Spring AOP中，有三个非常专业术语- Advices, Yiibaicut , Advisor，把它在非官方的方式...

- Advice – 指示之前或方法执行后采取的行动。
- Yiibaicut – 指明哪些方法应该拦截，通过方法的名称或正则表达式模式。
- Advisor – 分组"通知"和"切入点"成为一个单元，并把它传递到代理工厂对象。

再次回顾上一个 Spring AOP通知的例子。

*File : CustomerService.java*

```
package com.yiibai.customer.services;

public class CustomerService
{
    private String name;
    private String url;

    public void setName(String name) {
        this.name = name;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public void printName(){
        System.out.println("Customer name : " + this.name);
    }

    public void printURL(){
        System.out.println("Customer website : " + this.url);
    }

    public void printThrowException(){
        throw new IllegalArgumentException();
    }

}
```

*File : applicationContext.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yong Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackAroundMethodBeanAdvice" class="com.yiibai.aop.H

    <bean id="customerServiceProxy"
                class="org.springframework.aop.framework.ProxyFacto

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>hijackAroundMethodBeanAdvice</value>
            </list>
        </property>
    </bean>
</beans>
```

*File : HijackAroundMethod.java*

```
package com.yiibai.aop;

import java.util.Arrays;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class HijackAroundMethod implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation methodInvocation) throws

        System.out.println("Method name : "
                + methodInvocation.getMethod().getName());
        System.out.println("Method arguments : "
                + Arrays.toString(methodInvocation.getArguments())

        System.out.println("HijackAroundMethod : Before method hija

        try {
            Object result = methodInvocation.proceed();
            System.out.println("HijackAroundMethod : Before after h

            return result;

        } catch (IllegalArgumentException e) {

            System.out.println("HijackAroundMethod : Throw exceptio
            throw e;
        }
    }
}
```

执行它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

import com.yiibai.customer.services.CustomerService;

public class App {
    public static void main(String[] args) {
        ApplicationContext appContext = new ClassPathXmlApplicatior
                new String[] { "applicationContext.xml" });

        CustomerService cust = (CustomerService) appContext
                .getBean("customerServiceProxy");

        System.out.println("*************************");
        cust.printName();
        System.out.println("*************************");
        cust.printURL();
        System.out.println("*************************");
        try {
            cust.printThrowException();
        } catch (Exception e) {
        }
    }
}
```

输出

```
*************************
Method name : printName
Method arguments : []
HijackAroundMethod : Before method hijacked!
Customer name : Yiibai
HijackAroundMethod : Before after hijacked!
*************************
Method name : printURL
Method arguments : []
HijackAroundMethod : Before method hijacked!
Customer website : http://www.yiibai.com
HijackAroundMethod : Before after hijacked!
*************************
Method name : printThrowException
Method arguments : []
HijackAroundMethod : Before method hijacked!
HijackAroundMethod : Throw exception hijacked!
```

客户服务类的全部方法被截获。后来，我们展示如何使用"切入点"只拦截printName()方法。

# 切入点的例子

可以通过以下两种方式相匹配的方法：

1. 名称匹配
2. 正则表达式匹配

# 1.切入点 - 名称匹配的例子

通过“切入点”和“advisor”拦截printName()方法。创建NameMatchMethodYiibaicut切入点bean，并提出要在“mappedName”属性值来拦截方法名。

```
<bean id="customerYiibaicut"
        class="org.springframework.aop.support.NameMatchMethodYiiba
        <property name="mappedName" value="printName" />
    </bean>
```

创建 DefaultYiibaicutAdvisor 通知 bean，通知和切入点相关联。

```
<bean id="customerAdvisor"
        class="org.springframework.aop.support.DefaultYiibaicutAdvi
        <property name="pointcut" ref="customerYiibaicut" />
        <property name="advice" ref="hijackAroundMethodBeanAdvice"
    </bean>
```

更换代理“interceptorNames”到“customerAdvisor”（它是“hijackAroundMethodBeanAdvice”）。

```
<bean id="customerServiceProxy"
        class="org.springframework.aop.framework.ProxyFactoryBean">

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>customerAdvisor</value>
            </list>
        </property>
    </bean>
```

全部bean配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yiibai" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackAroundMethodBeanAdvice" class="com.yiibai.aop.H

    <bean id="customerServiceProxy"
                class="org.springframework.aop.framework.ProxyFacto

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>customerAdvisor</value>
            </list>
        </property>
    </bean>

    <bean id="customerYiibaicut"
                class="org.springframework.aop.support.NameMatchMet
        <property name="mappedName" value="printName" />
    </bean>

    <bean id="customerAdvisor"
                 class="org.springframework.aop.support.DefaultYiib
        <property name="pointcut" ref="customerYiibaicut" />
        <property name="advice" ref="hijackAroundMethodBeanAdvice"
    </bean>

</beans>
```

再次运行，输出

```
*************************
Method name : printName
Method arguments : []
HijackAroundMethod : Before method hijacked!
Customer name : Yiibai
HijackAroundMethod : Before after hijacked!
*************************
Customer website : http://www.yiibai.com
*************************
```

现在，只拦截 printName()方法。**YiibaicutAdvisor** Spring提供了YiibaicutAdvisor类来保存工作声明advisor和切入点到不同的bean，可以使用NameMatchMethodYiibaicutAdvisor 两者结合成一个 bean。

```
<bean id="customerAdvisor"
        class="org.springframework.aop.support.NameMatchMethodYiiba

        <property name="mappedName" value="printName" />
        <property name="advice" ref="hijackAroundMethodBeanAdvice"

    </bean>
```

## 2.切入点 - 正则表达式的例子

也可以通过使用正则表达式匹配切入点方法的名称 – RegexpMethodYiibaicutAdvisor.

```
<bean id="customerAdvisor"
        class="org.springframework.aop.support.RegexpMethodYiibaicu
        <property name="patterns">
            <list>
                <value>.*URL.*</value>
            </list>
        </property>

        <property name="advice" ref="hijackAroundMethodBeanAdvice"
    </bean>
```

现在，它拦截方法名称中有“URL”的方法。在实践中，可以用它来管理DAO层，声明“*.DAO.*” 拦截所有的DAO类来支持事务。

下载代码 – http://pan.baidu.com/s/1pJU2PoJ

# Spring自动代理创建者实例 - Spring教程

在上一篇 Spring AOP实例 – advice, pointcut 和 advisor, 必须手动创建一个代理bean(ProxyFactryBean)，对每个Bean需要AOP支持。这不是一种有效的方式，例如，如果想在客户模块，所有的DAO类实现SQL日志支持(提醒)的AOP功能，那么必须手动创建很多代理工厂bean，因此在 bean配置文件可能会泛滥代理类。幸运的是，Spring有两个自动代理创建者来自动创建代理bean。

## 1. BeanNameAutoProxyCreator示例

在此之前，必须手动创建一个代理bean(ProxyFactryBean)。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yiibai Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackAroundMethodBeanAdvice" class="com.yiibai.aop.H

    <bean id="customerServiceProxy"
        class="org.springframework.aop.framework.ProxyFactoryBean">

        <property name="target" ref="customerService" />

        <property name="interceptorNames">
            <list>
                <value>customerAdvisor</value>
            </list>
        </property>
    </bean>

    <bean id="customerAdvisor"
        class="org.springframework.aop.support.NameMatchMethodTutor
        <property name="mappedName" value="printName" />
        <property name="advice" ref="hijackAroundMethodBeanAdvice"
    </bean>
</beans>
```

并代理名称“customerServiceProxy”获得 bean。

```
CustomerService cust = (CustomerService)appContext.getBean("custome
```

在自动代理机制，只需要创建一个的BeanNameAutoProxyCreator，并包含所有你的bean(通过bean的名字，或正则表达式名)和“advisor” 作为一个单位。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yiibai Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackAroundMethodBeanAdvice" class="com.yiibai.aop.H

    <bean id="customerAdvisor"
        class="org.springframework.aop.support.NameMatchMethodTutor
        <property name="mappedName" value="printName" />
        <property name="advice" ref="hijackAroundMethodBeanAdvice"
    </bean>

    <bean
        class="org.springframework.aop.framework.autoproxy.BeanName
        <property name="beanNames">
            <list>
                <value>*Service</value>
            </list>
        </property>
        <property name="interceptorNames">
            <list>
                <value>customerAdvisor</value>
            </list>
        </property>
    </bean>
</beans>
```

现在，可以通过“CustomerService”的原始名称获取bean, 如果知道这个bean已经代理。

```
CustomerService cust = (CustomerService)appContext.getBean("custome
```

## 2. DefaultAdvisorAutoProxyCreator示例

这个 DefaultAdvisorAutoProxyCreator 是非常强大的，如果有任何的 bean 可相提并论，Spring会自动创建一个代理。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.

    <bean id="customerService" class="com.yiibai.customer.services.
        <property name="name" value="Yiibai Mook Kim" />
        <property name="url" value="http://www.yiibai.com" />
    </bean>

    <bean id="hijackAroundMethodBeanAdvice" class="com.yiibai.aop.H

    <bean id="customerAdvisor"
        class="org.springframework.aop.support.NameMatchMethodTutor
        <property name="mappedName" value="printName" />
        <property name="advice" ref="hijackAroundMethodBeanAdvice"
    </bean>

       <bean class="org.springframework.aop.framework.autoproxy.Def

</beans>
```

这仅仅是对权力，因为你不用管什么 Bean 应该被代理，可以做的就是相信Spring会做最适合你的。如果要实现这个到你的项目，请妥善保管。下载代码 – http://pan.baidu.com/s/1pKdqtjt

# Spring AOP+AspectJ注解实例 - Spring教程

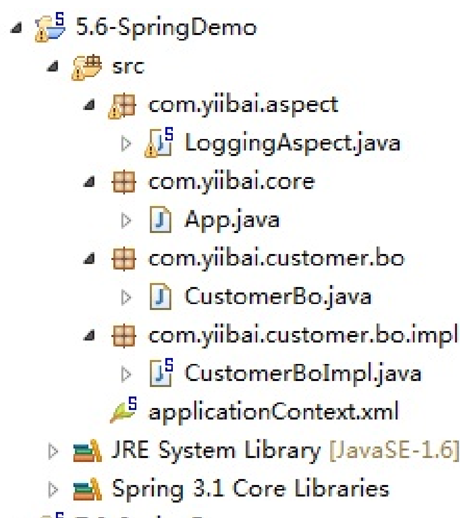
在本教程中，我们将向你展示如何将AspectJ注解集成到Spring AOP框架。在这个Spring AOP+ AspectJ 示例中，让您轻松实现拦截方法。常见AspectJ的注解：

1. @Before – 方法执行前运行
2. @After – 运行在方法返回结果后
3. @AfterReturning – 运行在方法返回一个结果后，在拦截器返回结果。
4. @AfterThrowing – 运行方法在抛出异常后，
5. @Around – 围绕方法执行运行，结合以上这三个通知。

注意 Spring AOP 中没有 AspectJ 支持，请阅读 内置 Spring AOP 例子。

## 1. 目录结构

看到这个例子的目录结构。




## 2. Spring Beans

普通 bean 中有几个方法，后来通过 AspectJ 注解拦截。

```
package com.yiibai.customer.bo;

public interface CustomerBo {

    void addCustomer();

    String addCustomerReturnValue();

    void addCustomerThrowException() throws Exception;

    void addCustomerAround(String name);
}
```

```
package com.yiibai.customer.bo.impl;

import com.yiibai.customer.bo.CustomerBo;

public class CustomerBoImpl implements CustomerBo {

    public void addCustomer(){
        System.out.println("addCustomer() is running ");
    }

    public String addCustomerReturnValue(){
        System.out.println("addCustomerReturnValue() is running ");
        return "abc";
    }

    public void addCustomerThrowException() throws Exception {
        System.out.println("addCustomerThrowException() is running
        throw new Exception("Generic Error");
    }

    public void addCustomerAround(String name){
        System.out.println("addCustomerAround() is running, args :
    }
}
```

## 4. 启用AspectJ

在 Spring 配置文件，把“”，并定义Aspect(拦截)和普通的bean。

*File : applicationContext.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

    <aop:aspectj-autoproxy />

    <bean id="customerBo" class="com.yiibai.customer.bo.impl.Custor

    <!-- Aspect -->
    <bean id="logAspect" class="com.yiibai.aspect.LoggingAspect" />

</beans>
```

# 4. AspectJ @Before

在下面例子中，logBefore()方法将在 customerBo接口的 addCustomer()方法的执行之前被执行。AspectJ的“切入点”是用来声明哪种方法将被拦截，应该参考Spring AOP切入点指南，支持切入点表达式的完整列表。

*File : LoggingAspect.java*

```
package com.yiibai.aspect;

import org.aspectj.lang.JoinYiibai;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class LoggingAspect {

    @Before("execution(* com.yiibai.customer.bo.CustomerBo.addCusto
    public void logBefore(JoinYiibai joinYiibai) {

        System.out.println("logBefore() is running!");
        System.out.println("hijacked : " + joinYiibai.getSignature(
        System.out.println("******");
    }

}
```

运行

```
CustomerBo customer = (CustomerBo) appContext.getBean("customerBo"
customer.addCustomer();
```

输出结果

```
logBefore() is running!
hijacked : addCustomer
******
addCustomer() is running
```

# 5. AspectJ @After

在下面例子中，logAfter()方法将在 customerBo 接口的 addCustomer()方法的执行之后执行。

*File : LoggingAspect.java*

```
package com.yiibai.aspect;

import org.aspectj.lang.JoinYiibai;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.After;

@Aspect
public class LoggingAspect {

    @After("execution(* com.yiibai.customer.bo.CustomerBo.addCustom
    public void logAfter(JoinYiibai joinYiibai) {

        System.out.println("logAfter() is running!");
        System.out.println("hijacked : " + joinYiibai.getSignature(
        System.out.println("******");

    }

}
```

运行它

```
CustomerBo customer = (CustomerBo) appContext.getBean("customerBo"
customer.addCustomer();
```

输出结果

```
addCustomer() is running
logAfter() is running!
hijacked : addCustomer
******
```

# 6. AspectJ @AfterReturning

在下面例子中，logAfterReturning()方法将在 customerBo 接口的 addCustomerReturnValue()方法执行之后执行。此外，还可以截取返回的值使用“returning”属性。

要截取返回的值，对“returning”属性(结果)的值必须用相同的方法参数(结果)。

*File : LoggingAspect.java*

```
package com.yiibai.aspect;

import org.aspectj.lang.JoinYiibai;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.AfterReturning;

@Aspect
public class LoggingAspect {

   @AfterReturning(
      pointcut = "execution(* com.yiibai.customer.bo.CustomerBo.adc
      returning= "result")
   public void logAfterReturning(JoinYiibai joinYiibai, Object resu

    System.out.println("logAfterReturning() is running!");
    System.out.println("hijacked : " + joinYiibai.getSignature().ge
    System.out.println("Method returned value is : " + result);
    System.out.println("******");
   }
}
```

运行它

```
CustomerBo customer = (CustomerBo) appContext.getBean("customerBo")
    customer.addCustomerReturnValue();
```

输出结果

```
addCustomerReturnValue() is running
logAfterReturning() is running!
hijacked : addCustomerReturnValue
Method returned value is : abc
******
```

# 7. AspectJ @AfterReturning

在下面的例子中，如果 customerBo 接口的addCustomerThrowException()方法抛出异常logAfterThrowing()方法将被执行。

*File : LoggingAspect.java*

```
package com.yiibai.aspect;

import org.aspectj.lang.JoinYiibai;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.AfterThrowing;

@Aspect
public class LoggingAspect {

   @AfterThrowing(
      pointcut = "execution(* com.yiibai.customer.bo.CustomerBo.add
      throwing= "error")
    public void logAfterThrowing(JoinYiibai joinYiibai, Throwable e

    System.out.println("logAfterThrowing() is running!");
    System.out.println("hijacked : " + joinYiibai.getSignature().ge
    System.out.println("Exception : " + error);
    System.out.println("******");

    }
}
```

运行它

```
CustomerBo customer = (CustomerBo) appContext.getBean("customerBo"
customer.addCustomerThrowException();
```

输出结果

```
addCustomerThrowException() is running
logAfterThrowing() is running!
hijacked : addCustomerThrowException
Exception : java.lang.Exception: Generic Error
******
Exception in thread "main" java.lang.Exception: Generic Error
    //...
```

# 8. AspectJ @Around

在下面例子中，logAround()方法将在customerBo接口的addCustomerAround()方法执行之前执行， 必须定义“joinYiibai.proceed();” 控制何时拦截器返回控制到原来的addCustomerAround()方法。

*File : LoggingAspect.java*

```
package com.yiibai.aspect;

import org.aspectj.lang.ProceedingJoinYiibai;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Around;

@Aspect
public class LoggingAspect {

   @Around("execution(* com.yiibai.customer.bo.CustomerBo.addCustom
   public void logAround(ProceedingJoinYiibai joinYiibai) throws Th

    System.out.println("logAround() is running!");
    System.out.println("hijacked method : " + joinYiibai.getSignatu
    System.out.println("hijacked arguments : " + Arrays.toString(jo

    System.out.println("Around before is running!");
    joinYiibai.proceed(); //continue on the intercepted method
    System.out.println("Around after is running!");

    System.out.println("******");

   }

}
```

运行它

```
CustomerBo customer = (CustomerBo) appContext.getBean("customerBo")
customer.addCustomerAround("yiibai");
```

输出结果

```
logAround() is running!
hijacked method : addCustomerAround
hijacked arguments : [yiibai]
Around before is running!
addCustomerAround() is running, args : yiibai
Around after is running!
******
```

# 总结

它总是建议采用最少 AspectJ 注解。这是关于Spring AspectJ 的一篇相当长的文章。进一步的解释和例子，请访问下面的参考链接。下载源代码 – http://pan.baidu.com/s/1boo4f9P

# Spring Object到XML映射实例 - Spring教程

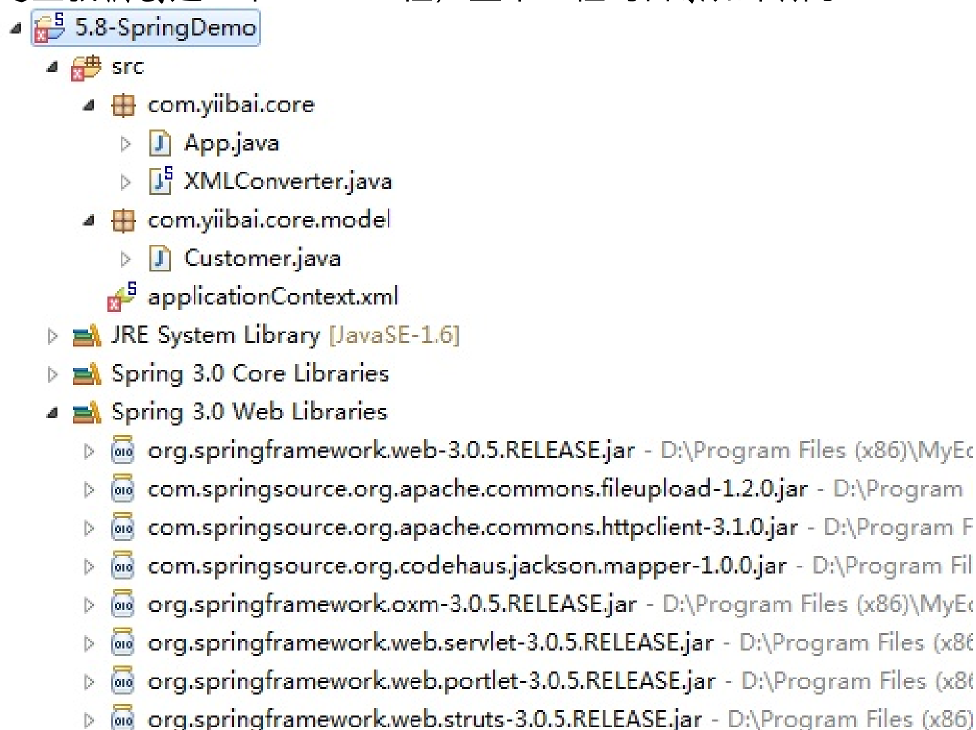
Spring的Object/XML映射将对象转换到XML，或反之亦然。这个过程也被称为

1. XML Marshalling – 转换对象到XML
2. XML UnMarshalling – 转换XML到对象

在本教程中，我们将介绍如何使用 Spring 的 OXM 来做转换， Object <--- Spring oxm ---> XML.

## 注: 为什么使用 Spring的OXM 有好处？请阅读本 Spring 对象/XML映射的文章。

这里我们创建一个 Java 工程，整个工程的目录如下所示：




## 1. 一个简单对象

一个简单的对象，之后将其转换成 XML 文件。

```
package com.yiibai.core.model;

public class Customer {

    String name;
    int age;
    boolean flag;
    String address;

    //standard getter, setter and toString() methods.
}
```

# 3. Marshaller 和 Unmarshaller

这个类将处理通过 Spring 的 OXM 接口的转换： Marshaller 和 Unmarshaller.

```
package com.yiibai.core;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
import org.springframework.oxm.Marshaller;
import org.springframework.oxm.Unmarshaller;

public class XMLConverter {

    private Marshaller marshaller;
    private Unmarshaller unmarshaller;

    public Marshaller getMarshaller() {
        return marshaller;
    }

    public void setMarshaller(Marshaller marshaller) {
        this.marshaller = marshaller;
    }

    public Unmarshaller getUnmarshaller() {
        return unmarshaller;
    }

    public void setUnmarshaller(Unmarshaller unmarshaller) {
        this.unmarshaller = unmarshaller;
    }

    public void convertFromObjectToXML(Object object, String filepa
        throws IOException {
```

```
        FileOutputStream os = null;
        try {
            os = new FileOutputStream(filepath);
            getMarshaller().marshal(object, new StreamResult(os));
        } finally {
            if (os != null) {
                os.close();
            }
        }
    }

    public Object convertFromXMLToObject(String xmlfile) throws IOE

        FileInputStream is = null;
        try {
            is = new FileInputStream(xmlfile);
            return getUnmarshaller().unmarshal(new StreamSource(is)
        } finally {
            if (is != null) {
                is.close();
            }
        }
    }

}
```

## 4. Spring配置

在 Spring 的 bean 配置文件，注入 CastorMarshaller 作为 XML 绑定框架。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="XMLConverter" class="com.yiibai.core.XMLConverter">
        <property name="marshaller" ref="castorMarshaller" />
        <property name="unmarshaller" ref="castorMarshaller" />
    </bean>
    <bean id="castorMarshaller" class="org.springframework.oxm.cast

</beans>
```

## 5. 测试

运行它

```
package com.yiibai.core;

import java.io.IOException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(
import com.yiibai.core.model.Customer;

public class App {
    private static final String XML_FILE_NAME = "customer.xml";

    public static void main(String[] args) throws IOException {
        ApplicationContext appContext = new ClassPathXmlApplicatio
        XMLConverter converter = (XMLConverter) appContext.getBean(

        Customer customer = new Customer();
        customer.setName("yiibai");
        customer.setAge(28);
        customer.setFlag(true);
        customer.setAddress("Haikou haidiandao");

        System.out.println("Convert Object to XML!");
        //from object to XML file
        converter.convertFromObjectToXML(customer, XML_FILE_NAME);
        System.out.println("Done \n");

        System.out.println("Convert XML back to Object!");
        //from XML to object
        Customer customer2 = (Customer)converter.convertFromXMLToOb
        System.out.println(customer2);
        System.out.println("Done");

    }
}
```

输出结果

```
Convert Object to XML!
Done

Convert XML back to Object!
Customer [name=yiibai, age=28, flag=true, address=Haikou Haidianda
Done
```

下面的 XML 文件“customer.xml”将在项目的根文件夹中生成。

*File : customer.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<customer flag="true" age="28">
    <address>Haikou Haidiandao</address>
    <name>yiibai</name>
</customer>
```

## XML映射

等等，为什么flag和age可转换为属性？这是一种来控制哪些字段应为属性或元素的使用的方式？ 当然，您可以使用 Castor XML映射定义对象 和XML之间的关系。

创建以下映射文件，并把它放到你的项目的 classpath。

*File : mapping.xml*

```
<mapping>
    <class name="com.yiibai.core.model.Customer">

        <map-to xml="customer" />

        <field name="age" type="integer">
            <bind-xml name="age" node="attribute" />
        </field>

        <field name="flag" type="boolean">
            <bind-xml name="flag" node="element" />
        </field>

        <field name="name" type="string">
            <bind-xml name="name" node="element" />
        </field>

        <field name="address" type="string">
            <bind-xml name="address" node="element" />
        </field>
    </class>
</mapping>
```

在Spring bean配置文件，上述通过“mappingLocation”注入 mapping.xml 到 CastorMarshaller 。注：这里需要加入一个 org.springframework.oxm.*.jar 包，这个包函数在 MyEclipse 库的 Spring 3.0 Web Libaries中。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xs

    <bean id="XMLConverter" class="com.yiibai.core.XMLConverter">
        <property name="marshaller" ref="castorMarshaller" />
        <property name="unmarshaller" ref="castorMarshaller" />
    </bean>
    <bean id="castorMarshaller" class="org.springframework.oxm.cast
        <property name="mappingLocation" value="classpath:mapping.x
    </bean>

</beans>
```

再次测试，XML文件“customer.xml”将被更新。

*File : customer.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<customer age="28">
    <flag>true</flag>
    <name>yiibai</name>
    <address>Haikou Haidiandao</address>
</customer>
```

参考：http://my.oschina.net/u/1455908/blog/311723下载文档 – http://pan.baidu.com/s/1dEo0bnz

# Spring+JDBC实例 - Spring教程

## 1. Customer 表

在这个例子中，我们使用的是MySQL数据库。

```
CREATE TABLE `customer` (
  `CUST_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `NAME` varchar(100) NOT NULL,
  `AGE` int(10) unsigned NOT NULL,
  PRIMARY KEY (`CUST_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

## 2. Customer模型

添加一个客户模型用来存储用户的数据。

```
package com.yiibai.customer.model;

import java.sql.Timestamp;

public class Customer
{
    int custId;
    String name;
    int age;
    //getter and setter methods

}
```

## 4. 数据访问对象 (DAO) 模式

Customer Dao 接口.

```
package com.yiibai.customer.dao;

import com.yiibai.customer.model.Customer;

public interface CustomerDAO
{
    public void insert(Customer customer);
    public Customer findByCustomerId(int custId);
}
```

客户的DAO实现，使用 JDBC 发出简单的 insert 和 select SQL语句。

```
package com.yiibai.customer.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.model.Customer;

public class JdbcCustomerDAO implements CustomerDAO
{
    private DataSource dataSource;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void insert(Customer customer){

        String sql = "INSERT INTO CUSTOMER " +
                "(CUST_ID, NAME, AGE) VALUES (?, ?, ?)";
        Connection conn = null;

        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, customer.getCustId());
            ps.setString(2, customer.getName());
            ps.setInt(3, customer.getAge());
            ps.executeUpdate();
            ps.close();

        } catch (SQLException e) {
            throw new RuntimeException(e);

        } finally {
            if (conn != null) {
                try {
```

```
                    conn.close();
                } catch (SQLException e) {}
            }
        }
    }

    public Customer findByCustomerId(int custId){

        String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";

        Connection conn = null;

        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, custId);
            Customer customer = null;
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                customer = new Customer(
                    rs.getInt("CUST_ID"),
                    rs.getString("NAME"),
                    rs.getInt("Age")
                );
            }
            rs.close();
            ps.close();
            return customer;
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            if (conn != null) {
                try {
                conn.close();
                } catch (SQLException e) {}
            }
        }
    }
}
```

# 5. Spring bean配置

创建 customerDAO 和数据源在 Spring bean 配置文件中。

*File : Spring-Customer.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="customerDAO" class="com.yiibai.customer.dao.impl.Jdbc
        <property name="dataSource" ref="dataSource" />
    </bean>

</beans>
```

*File : Spring-Datasource.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDat

        <property name="driverClassName" value="com.mysql.jdbc.Driv
        <property name="url" value="jdbc:mysql://localhost:3306/yii
        <property name="username" value="root" />
        <property name="password" value="password" />
    </bean>

</beans>
```

*File : Spring-Module.xml*

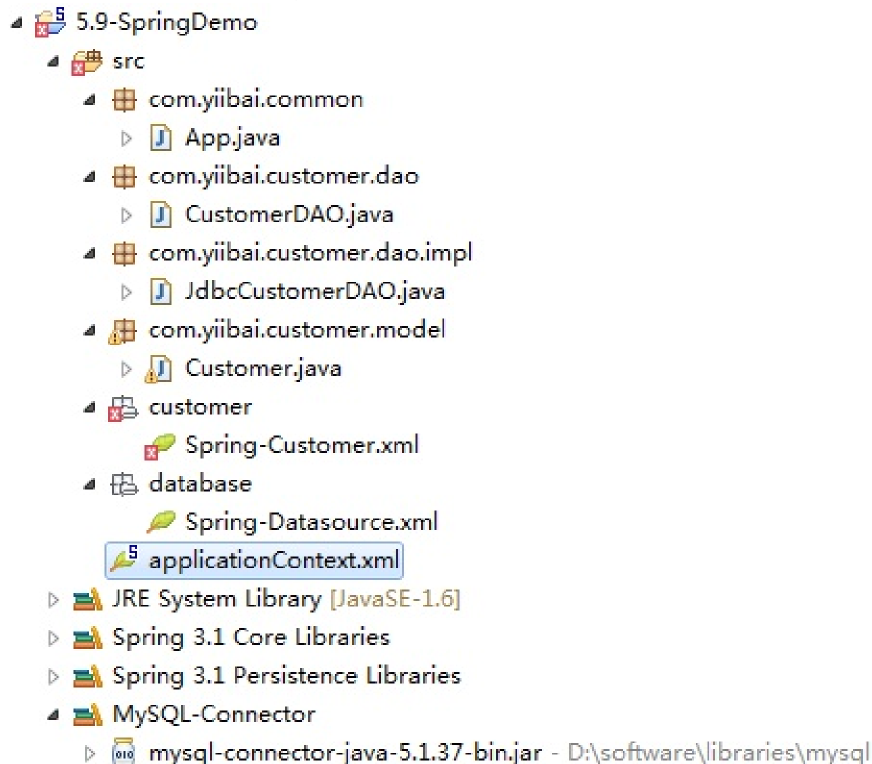
```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <import resource="database/Spring-Datasource.xml" />
    <import resource="customer/Spring-Customer.xml" />

</beans>
```

# 6.项目结构

本实例完整目录结构。




# 7.运行它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.model.Customer;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.

        CustomerDAO customerDAO = (CustomerDAO) context.getBean("cu
        Customer customer = new Customer(1, "yiibai",29);
        customerDAO.insert(customer);

        Customer customer1 = customerDAO.findByCustomerId(1);
        System.out.println(customer1);

    }
}
```

输出结果：

```
Customer [age=29, custId=1, name=yiibai]
```

下载源代码 – http://pan.baidu.com/s/1SR2GI

# Spring JdbcTemplate+JdbcDaoSupport实例 - Spring教程

在Spring JDBC开发中，可以使用 JdbcTemplate 和 JdbcDaoSupport 类来简化整个数据库的操作过程。在本教程中，我们将重复上一篇文章 Spring+JDBC例子，看之前(无JdbcTemplate支持)和之后(含JdbcTemplate的支持)之间不同的例子。

## 1. 不使用JdbcTemplate示例

如果不用JdbcTemplate，必须创建大量的冗余代码(创建连接，关闭连接，处理异常)中的所有DAO数据库的操作方法 - 插入，更新和删除。它的效率并不是很高，容易出错和乏味。

```
private DataSource dataSource;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void insert(Customer customer){

        String sql = "INSERT INTO CUSTOMER " +
                "(CUST_ID, NAME, AGE) VALUES (?, ?, ?)";
        Connection conn = null;

        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, customer.getCustId());
            ps.setString(2, customer.getName());
            ps.setInt(3, customer.getAge());
            ps.executeUpdate();
            ps.close();

        } catch (SQLException e) {
            throw new RuntimeException(e);

        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {}
            }
        }
    }
```

## 2. 使用JdbcTemplate示例

使用JdbcTemplate可节省大量的冗余代码，因为JdbcTemplate类会自动处理它。

```
private DataSource dataSource;
    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void insert(Customer customer){

        String sql = "INSERT INTO CUSTOMER " +
            "(CUST_ID, NAME, AGE) VALUES (?, ?, ?)";

        jdbcTemplate = new JdbcTemplate(dataSource);

        jdbcTemplate.update(sql, new Object[] { customer.getCustId(
            customer.getName(),customer.getAge()
        });

    }
```

看看有什么不同？

## 3. 使用JdbcDaoSupport示例

通过扩展 JdbcDaoSupport，设置数据源，并且 JdbcTemplate 在你的类中不再是必需的，只需要正确的数据源注入JdbcCustomerDAO。可以使用getJdbcTemplate()方法得到 JdbcTemplate。

```
public class JdbcCustomerDAO extends JdbcDaoSupport implements Cust
    {
       //no need to set datasource here
       public void insert(Customer customer){

        String sql = "INSERT INTO CUSTOMER " +
            "(CUST_ID, NAME, AGE) VALUES (?, ?, ?)";

        getJdbcTemplate().update(sql, new Object[] { customer.getCu
                customer.getName(),customer.getAge()
        });

    }
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="dataSource"
         class="org.springframework.jdbc.datasource.DriverManagerDa

        <property name="driverClassName" value="com.mysql.jdbc.Driv
        <property name="url" value="jdbc:mysql://localhost:3306/yi
        <property name="username" value="root" />
        <property name="password" value="password" />
    </bean>

</beans>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="customerDAO" class="com.yiibai.customer.dao.impl.Jdbc
        <property name="dataSource" ref="dataSource" />
    </bean>

</beans>
```

注: 在Spring JDBC开发，它总是建议使用，总是建议使用 JdbcTemplate和JdbcDaoSupport，而不使用自己的JDBC编程代码。下载代码 – http://pan.baidu.com/s/1bnIGdiR

# Spring JdbcTemplate查询实例 - Spring教程

这里有几个例子向您展示如何使用JdbcTemplate的query()方法来查询或从数据库提取数据。整个项目的目录结构如下：

## 1.查询单行数据

这里有两种方法来查询或从数据库中提取单行记录，并将其转换成一个模型类。

## 1.1 自定义RowMapper

在一般情况下，它总是建议实现 RowMapper 接口来创建自定义的RowMapper，以满足您的需求。

```
package com.yiibai.customer.model;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class CustomerRowMapper implements RowMapper
{
    public Object mapRow(ResultSet rs, int rowNum) throws SQLExcept
        Customer customer = new Customer();
        customer.setCustId(rs.getInt("CUST_ID"));
        customer.setName(rs.getString("NAME"));
        customer.setAge(rs.getInt("AGE"));
        return customer;
    }

}
```

它传递给 queryForObject()方法，返回的结果将调用自定义 mapRow()方法的值匹配到属性。

```
public Customer findByCustomerId(int custId){

    String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";

    Customer customer = (Customer)getJdbcTemplate().queryForObject(
            sql, new Object[] { custId }, new CustomerRowMapper());

    return customer;
}
```

## 1.2 BeanPropertyRowMapper

在Spring2.5中，带有一个方便 RowMapper 实现所谓“BeanPropertyRowMapper”，它可以通过匹配行的名字的列值映射到一个属性。只要确保这两个属性和列具有相同的名称，如属性“CUSTID'将匹配到列名为：”CUSTID'或下划线“CUST_ID”。

```
public Customer findByCustomerId2(int custId){

    String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";

    Customer customer = (Customer)getJdbcTemplate().queryForObject(
            sql, new Object[] { custId },
            new BeanPropertyRowMapper(Customer.class));

    return customer;
}
```

## 2，查询多行

现在，查询或从数据库中提取多行，并且将它转换成一个列表。

## 2.1手动映射它

返回多行，RowMapper 不支持 queryForList()方法，需要手动映射它。

```
public List<Customer> findAll(){

    String sql = "SELECT * FROM CUSTOMER";

    List<Customer> customers = new ArrayList<Customer>();

    List<Map> rows = getJdbcTemplate().queryForList(sql);
    for (Map row : rows) {
        Customer customer = new Customer();
        customer.setCustId((Long)(row.get("CUST_ID")));
        customer.setName((String)row.get("NAME"));
        customer.setAge((Integer)row.get("AGE"));
        customers.add(customer);
    }

    return customers;
}
```

## 2.2 BeanPropertyRowMapper

最简单的解决方案是使用 BeanPropertyRowMapper 类。

```
public List<Customer> findAll(){

    String sql = "SELECT * FROM CUSTOMER";

    List<Customer> customers  = getJdbcTemplate().query(sql,
            new BeanPropertyRowMapper(Customer.class));

    return customers;
}
```

# 3.查询单值

在这个例子中，展示了如何从数据库中查询或提取单个列值。

## 3.1单列名

它显示了如何查询单个列名作为字符串。

```
public String findCustomerNameById(int custId){

    String sql = "SELECT NAME FROM CUSTOMER WHERE CUST_ID = ?";

    String name = (String)getJdbcTemplate().queryForObject(
            sql, new Object[] { custId }, String.class);

    return name;

}
```

## 3.2、行总数

它展示了如何从数据库中查询行的总数。

```
public int findTotalCustomer(){

    String sql = "SELECT COUNT(*) FROM CUSTOMER";

    int total = getJdbcTemplate().queryForInt(sql);

    return total;
}
```

运行它

```
package com.yiibai.common;

import java.util.ArrayList;
import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.model.Customer;

public class JdbcTemplateApp
{
    public static void main( String[] args )
    {
         ApplicationContext context =
            new ClassPathXmlApplicationContext("Spring-Customer.xml
	
         CustomerDAO customerDAO = (CustomerDAO) context.getBean("c

         Customer customerA = customerDAO.findByCustomerId(1);
         System.out.println("Customer A : " + customerA);

         Customer customerB = customerDAO.findByCustomerId2(1);
         System.out.println("Customer B : " + customerB);

         List<Customer> customerAs = customerDAO.findAll();
         for(Customer cust: customerAs){
              System.out.println("Customer As : " + customerAs);
         }

         List<Customer> customerBs = customerDAO.findAll2();
         for(Customer cust: customerBs){
              System.out.println("Customer Bs : " + customerBs);
         }

         String customerName = customerDAO.findCustomerNameById(1);
         System.out.println("Customer Name : " + customerName);

         int total = customerDAO.findTotalCustomer();
         System.out.println("Total : " + total);

    }
}
```

# 总结

JdbcTemplate类，附带了很多有用的重载查询方法。它提醒参考现有的查询方法在创建自己的自定义查询方法之前，因为 Spring 已经做给你了。下载代码 – http://pan.baidu.com/s/1gecQHmN

# Spring SimpleJdbcTemplate查询示例 - Spring教程

这里有几个例子来说明如何使用SimpleJdbcTemplate query()方法来查询或从数据库中提取数据。在 JdbcTemplate query() 方法，需要手动转换返回的结果转换为一个目标对象类型，并传递一个对象数组作为参数。在SimpleJdbcTemplate类，它是更加人性化和简单。jdbctemplate VS simplejdbctemplate 请比较 JdbcTemplate类的示例和SimpleJdbcTemplate类的示例。

## 1.查询单行

这里有向你展示了如何查询或从数据库中提取单行的两种方式，并将其转换成一个模型类。

## 1.1 自定义RowMapper

在一般情况下，它总是建议来实现 RowMapper 接口来创建自定义的RowMapper，以满足您的需求。

```
package com.yiibai.customer.model;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class CustomerRowMapper implements RowMapper
{
    public Object mapRow(ResultSet rs, int rowNum) throws SQLExcept
        Customer customer = new Customer();
        customer.setCustId(rs.getInt("CUST_ID"));
        customer.setName(rs.getString("NAME"));
        customer.setAge(rs.getInt("AGE"));
        return customer;
    }

}
```

```
public Customer findByCustomerId(int custId){

    String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";

    Customer customer = getSimpleJdbcTemplate().queryForObject(
            sql,  new CustomerParameterizedRowMapper(), custId);

    return customer;
}
```

## 1.2 BeanPropertyRowMapper

在SimpleJdbcTemplate类，需要使用"ParameterizedBeanPropertyRowMapper' 代替 'BeanPropertyRowMapper"。

```
public Customer findByCustomerId2(int custId){

    String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";

    Customer customer = getSimpleJdbcTemplate().queryForObject(sql,
          ParameterizedBeanPropertyRowMapper.newInstance(Customer.c

    return customer;
}
```

## 2，查询多行

从数据库查询或提取多行记录，并将其转换成一个列表。

## 2.1 ParameterizedBeanPropertyRowMapper

```
public List<Customer> findAll(){

    String sql = "SELECT * FROM CUSTOMER";

    List<Customer> customers =
        getSimpleJdbcTemplate().query(sql,
           ParameterizedBeanPropertyRowMapper.newInstance(Customer.

    return customers;
}
```

# 3.查询单值

查询或提取数据库中的单个列的值。

## 3.1单列名

它显示了如何查询单个列名作为字符串。

```
public String findCustomerNameById(int custId){

    String sql = "SELECT NAME FROM CUSTOMER WHERE CUST_ID = ?";

    String name = getSimpleJdbcTemplate().queryForObject(
        sql, String.class, custId);

    return name;

}
```

## 3.2、行总数

它展示了如何从数据库中查询行的总数。

```
public int findTotalCustomer(){

    String sql = "SELECT COUNT(*) FROM CUSTOMER";

    int total = getSimpleJdbcTemplate().queryForInt(sql);

    return total;
}
```

运行它

```
package com.yiibai.common;

import java.util.ArrayList;
import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.model.Customer;

public class SimpleJdbcTemplateApp
{
    public static void main( String[] args )
    {
         ApplicationContext context =
            new ClassPathXmlApplicationContext("Spring-Customer.xml

         CustomerDAO customerSimpleDAO =
                (CustomerDAO) context.getBean("customerSimpleDAO");

         Customer customerA = customerSimpleDAO.findByCustomerId(1)
         System.out.println("Customer A : " + customerA);

         Customer customerB = customerSimpleDAO.findByCustomerId2(1
         System.out.println("Customer B : " + customerB);

         List<Customer> customerAs = customerSimpleDAO.findAll();
         for(Customer cust: customerAs){
              System.out.println("Customer As : " + customerAs);
         }

         List<Customer> customerBs = customerSimpleDAO.findAll2();
         for(Customer cust: customerBs){
              System.out.println("Customer Bs : " + customerBs);
         }

         String customerName = customerSimpleDAO.findCustomerNameBy
         System.out.println("Customer Name : " + customerName);

         int total = customerSimpleDAO.findTotalCustomer();
         System.out.println("Total : " + total);

    }
}
```

## 总结

SimpleJdbcTemplate 是不能代替 JdbcTemplate 的，它只是一个Java5的友好补充它。下载代码 – http://pan.baidu.com/s/1eRisz6M

# Spring SimpleJdbcTemplate类命名参数实例 - Spring教程

在JdbcTemplate，这些SQL参数通过一个特殊的占位符“？”符号表示，并通过位置绑定。现在的问题是，每当参数的顺序发生了变化，你必须也要改变参数绑定，这是容易出错，繁琐的维护。

为了解决这个问题，可以使用“命名参数”，SQL参数是由一个冒号开始后续定义的名称，而不是位置。在另外的，命名参数只是在SimpleJdbcTemplate类和NamedParameterJdbcTemplate支持。

请参见下面的三个例子用来使用命名参数在Spring。

## 示例1

例子向您展示如何使用命名参数在一个 INSERT 语句。

```
//insert with named parameter
    public void insertNamedParameter(Customer customer){

        String sql = "INSERT INTO CUSTOMER " +
            "(CUST_ID, NAME, AGE) VALUES (:custId, :name, :age)";

        Map<String, Object> parameters = new HashMap<String, Object
        parameters.put("custId", customer.getCustId());
        parameters.put("name", customer.getName());
        parameters.put("age", customer.getAge());

        getSimpleJdbcTemplate().update(sql, parameters);

    }
```

## 示例 2

例子来说明如何使用命名参数在批处理理操作语句。

```
public void insertBatchNamedParameter(final List<Customer> customer

        String sql = "INSERT INTO CUSTOMER " +
        "(CUST_ID, NAME, AGE) VALUES (:custId, :name, :age)";

        List<SqlParameterSource> parameters = new ArrayList<SqlPara
        for (Customer cust : customers) {
            parameters.add(new BeanPropertySqlParameterSource(cust)
        }

        getSimpleJdbcTemplate().batchUpdate(sql,
            parameters.toArray(new SqlParameterSource[0]));
    }
```

# 示例 3

另一个例子，在一个批处理操作语句中使用命名参数。

```
public void insertBatchNamedParameter2(final List<Customer> custome

       SqlParameterSource[] params =
        SqlParameterSourceUtils.createBatch(customers.toArray());

       getSimpleJdbcTemplate().batchUpdate(
        "INSERT INTO CUSTOMER (CUST_ID, NAME, AGE) VALUES (:custId,
        params);

    }
```

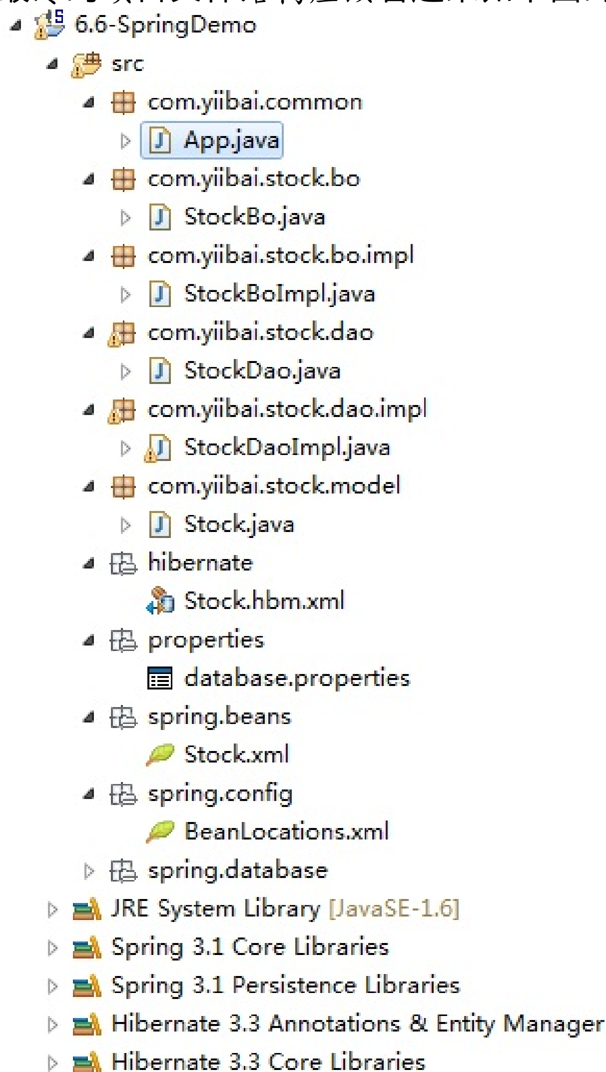
下载源代码 – http://pan.baidu.com/s/1hqPrzJu

# Spring+Hibernate+MySql实例 - Spring教程

这个例子将创建一个简单的 Java 项目结构，并演示如何使用 Hibernate 在Spring 框架进行 MySQL 数据库的数据处理工作(插入，选择，更新和删除)。

## 最终项目结构

最终的项目文件结构应该看起来如下面的图中显示。




## 1.创建表

在MySQL数据库中创建一张 “stock”表。 SQL语句如下：

```
CREATE TABLE  `yiibai`.`stock` (
  `STOCK_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `STOCK_CODE` varchar(10) NOT NULL,
  `STOCK_NAME` varchar(20) NOT NULL,
  PRIMARY KEY (`STOCK_ID`) USING BTREE,
  UNIQUE KEY `UNI_STOCK_NAME` (`STOCK_NAME`),
  UNIQUE KEY `UNI_STOCK_ID` (`STOCK_CODE`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

# 2. Model & BO & DAO

模型中，业务对象(BO)和数据访问对象(DAO)模式是有助于清楚地识别层，以避免弄乱项目结构。

### Stock Model

Stock 模型类以后用于存储库存数据。

```
package com.yiibai.stock.model;

import java.io.Serializable;

public class Stock implements Serializable {

    private static final long serialVersionUID = 1L;

    private Long stockId;
    private String stockCode;
    private String stockName;

    //getter and setter methods...
}
```

### Stock Business Object (BO))

Stock 业务对象(BO)接口和实现，它是用来存储项目的业务功能，真正的数据库操作(CRUD)的工作不应该参与这一个类，而是有一个DAO(StockDao)类来做到这一点。

```
package com.yiibai.stock.bo;

import com.yiibai.stock.model.Stock;

public interface StockBo {

    void save(Stock stock);
    void update(Stock stock);
    void delete(Stock stock);
    Stock findByStockCode(String stockCode);
}
```

```
package com.yiibai.stock.bo.impl;

import com.yiibai.stock.bo.StockBo;
import com.yiibai.stock.dao.StockDao;
import com.yiibai.stock.model.Stock;

public class StockBoImpl implements StockBo{

    StockDao stockDao;

    public void setStockDao(StockDao stockDao) {
        this.stockDao = stockDao;
    }

    public void save(Stock stock){
        stockDao.save(stock);
    }

    public void update(Stock stock){
        stockDao.update(stock);
    }

    public void delete(Stock stock){
        stockDao.delete(stock);
    }

    public Stock findByStockCode(String stockCode){
        return stockDao.findByStockCode(stockCode);
    }
}
```

### Stock Data Access Object

Stock DAO接口和实现，DAO实现类扩展了 Spring 的“HibernateDaoSupport”，以使Spring框架支持Hibernate。 现在，你可以通过getHibernateTemplate()执行Hibernate 功能。

```
package com.yiibai.stock.dao;

import com.yiibai.stock.model.Stock;

public interface StockDao {

    void save(Stock stock);
    void update(Stock stock);
    void delete(Stock stock);
    Stock findByStockCode(String stockCode);

}
```

```
package com.yiibai.stock.dao.impl;

import java.util.List;

import org.springframework.orm.hibernate3.support.HibernateDaoSuppo

import com.yiibai.stock.dao.StockDao;
import com.yiibai.stock.model.Stock;

public class StockDaoImpl extends HibernateDaoSupport implements St

    public void save(Stock stock){
        getHibernateTemplate().save(stock);
    }

    public void update(Stock stock){
        getHibernateTemplate().update(stock);
    }

    public void delete(Stock stock){
        getHibernateTemplate().delete(stock);
    }

    public Stock findByStockCode(String stockCode){
        List list = getHibernateTemplate().find(
                      "from Stock where stockCode=?",stockCode
                );
        return (Stock)list.get(0);
    }

}
```

## 5.资源配置

用于存储 Spring，Hibernate 等配置文件。

### Hibernate Configuration

创建 Hibernate 映射文件(Stock.hbm.xml)的 Stock 表，把它放在“resources/hibernate/”文件夹中。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.yiibai.stock.model.Stock" table="stock" catalo
        <id name="stockId" type="java.lang.Long">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">
            <column name="STOCK_CODE" length="10" not-null="true" u
        </property>
        <property name="stockName" type="string">
            <column name="STOCK_NAME" length="20" not-null="true" u
        </property>
    </class>
</hibernate-mapping>
```

### Spring Configuration

Database related....

创建一个属性文件(database.properties)数据库的详细信息，把它放在“resources/properties” 文件夹中。这是很好的做法，不同于数据库细节并将 Spring bean 配置成不同的文件。

### database.properties

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/yiibai
jdbc.username=root
jdbc.password=password
```

为你的数据库创建一个“dataSource” bean配置文件(DataSource.xml)，并从 database.properties导入的属性，把它放入到 “resources/database” 文件夹中。

### DataSource.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean
class="org.springframework.beans.factory.config.PropertyPlaceholde
    <property name="location">
        <value>properties/database.properties</value>
    </property>
</bean>

<bean id="dataSource"
         class="org.springframework.jdbc.datasource.DriverManagerDa
    <property name="driverClassName" value="${jdbc.driverClassName
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>

</beans>
```

Hibernate 关联….

创建一个会话工厂 bean 配置文件(Hibernate.xml)，把它放入 “resources/database” 文件夹中。这个 LocalSessionFactoryBean 中设置一个共享的 Hibernate SessionFactory 在一个 Spring 应用程序上下文。

**Hibernate.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<!-- Hibernate session factory -->
<bean id="sessionFactory"
     class="org.springframework.orm.hibernate3.LocalSessionFactoryB

    <property name="dataSource">
      <ref bean="dataSource"/>
    </property>

    <property name="hibernateProperties">
       <props>
         <prop key="hibernate.dialect">org.hibernate.dialect.MySQLD
         <prop key="hibernate.show_sql">true</prop>
       </props>
     </property>

     <property name="mappingResources">
    <list>
           <value>/hibernate/Stock.hbm.xml</value>
    </list>
      </property>

    </bean>
</beans>
```

Spring beans related….

创建一个bean配置文件(Stock.xml)的BO和DAO类，把它放入 “resources/spring” 文件夹中。依赖的 DAO(stockDao)bean 注入到 bo(stockBo)bean; SessionFactory 的 bean 成stockDao。

**Stock.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <!-- Stock business object -->
   <bean id="stockBo" class="com.yiibai.stock.bo.impl.StockBoImpl"
           <property name="stockDao" ref="stockDao" />
   </bean>

   <!-- Stock Data Access Object -->
   <bean id="stockDao" class="com.yiibai.stock.dao.impl.StockDaoImp
           <property name="sessionFactory" ref="sessionFactory"></p
   </bean>

</beans>
```

导入所有的Spring bean配置文件合并为一个文件(BeanLocations.xml)，把它变成了“resources/config”文件夹。

**BeanLocations.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <!-- Database Configuration -->
    <import resource="../database/DataSource.xml"/>
    <import resource="../database/Hibernate.xml"/>

    <!-- Beans Declaration -->
    <import resource="../beans/Stock.xml"/>

</beans>
```

# 6. 运行它

把所有的文件和配置，运行它。

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication(

import com.yiibai.stock.bo.StockBo;
import com.yiibai.stock.model.Stock;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext appContext =
          new ClassPathXmlApplicationContext("spring/config/BeanLoc

        StockBo stockBo = (StockBo)appContext.getBean("stockBo");

        /** insert **/
        Stock stock = new Stock();
        stock.setStockCode("7668");
        stock.setStockName("HAIO");
        stockBo.save(stock);

        /** select **/
        Stock stock2 = stockBo.findByStockCode("7668");
        System.out.println(stock2);

        /** update **/
        stock2.setStockName("HAIO-1");
        stockBo.update(stock2);

        /** delete **/
        stockBo.delete(stock2);

        System.out.println("Done");
    }
}
```

输出结果：

```
Hibernate: insert into yiibai.stock (STOCK_CODE, STOCK_NAME) values
Hibernate: select stock0_.STOCK_ID as STOCK1_0_, stock0_.STOCK_CODE
Stock [stockCode=7667, stockId=1, stockName=HAIO]
Hibernate: update yiibai.stock set STOCK_CODE=?, STOCK_NAME=? where
Done
```

下载代码 – http://pan.baidu.com/s/1i4f40tB

# Spring AOP在Hibernate事务管理 - Spring教程

事务管理是用来以确保数据库中数据的完整性和一致性。Spring AOP技术允许开发者管理事务的声明。这里有一个例子来说明如何使用Spring AOP 来管理 Hibernate 事务。整个工程的文件结构如下所示：

P.S 这里很多Hibernate和Spring配置文件是隐藏的，只有一些重要的文件显示，如果你想看全部文件，请在文章的结尾下载完整的项目代码。

## 1.创建表

MySQL表的脚本，一个“product”表

```
CREATE TABLE  `yiibai`.`product` (
  `PRODUCT_ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `PRODUCT_CODE` varchar(20) NOT NULL,
  `PRODUCT_DESC` varchar(255) NOT NULL,
  PRIMARY KEY (`PRODUCT_ID`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

CREATE TABLE  `yiibai`.`product_qoh` (
  `QOH_ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `PRODUCT_ID` bigint(20) unsigned NOT NULL,
  `QTY` int(10) unsigned NOT NULL,
  PRIMARY KEY (`QOH_ID`),
  KEY `FK_product_qoh_product_id` (`PRODUCT_ID`),
  CONSTRAINT `FK_product_qoh_product_id` FOREIGN KEY (`PRODUCT_ID`)
  REFERENCES `product` (`PRODUCT_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

## 2.产品业务对象

在这个“productBo”实现save()方法将插入记录到“product“通过“ProductDao”，并通过“productQohBo”类库存量到“productQoh'表。

```
package com.yiibai.product.bo.impl;

import com.yiibai.product.bo.ProductBo;
import com.yiibai.product.bo.ProductQohBo;
import com.yiibai.product.dao.ProductDao;
import com.yiibai.product.model.Product;
import com.yiibai.product.model.ProductQoh;

public class ProductBoImpl implements ProductBo{

    ProductDao productDao;
    ProductQohBo productQohBo;

    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }

    public void setProductQohBo(ProductQohBo productQohBo) {
        this.productQohBo = productQohBo;
    }

    //this method need to be transactional
    public void save(Product product, int qoh){

        productDao.save(product);
        System.out.println("Product Inserted");

        ProductQoh productQoh = new ProductQoh();
        productQoh.setProductId(product.getProductId());
        productQoh.setQty(qoh);

        productQohBo.save(productQoh);
        System.out.println("ProductQoh Inserted");
    }
}
```

Spring 的 bean 配置文件。

```
<!-- Product business object -->
   <bean id="productBo" class="com.yiibai.product.bo.impl.ProductBo
       <property name="productDao" ref="productDao" />
       <property name="productQohBo" ref="productQohBo" />
   </bean>

   <!-- Product Data Access Object -->
   <bean id="productDao" class="com.yiibai.product.dao.impl.Product
       <property name="sessionFactory" ref="sessionFactory"></prope
   </bean>
```

运行它

```
Product product = new Product();
    product.setProductCode("ABC");
    product.setProductDesc("This is product ABC");

    ProductBo productBo = (ProductBo)appContext.getBean("productBo'
    productBo.save(product, 100);
```

假设save() 不具有事务功能，如果异常抛出由productQohBo.save()，钭只插入一条记录到“product”表，没有记录将被插入到“productQoh'表。这是一个严重的问题，在数据库中打破数据一致性。

# 3.事务管理

声明“TransactionInterceptor' bean，以及”HibernateTransactionManager' Hibernate事务，并且通过必要的属性。

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="transactionInterceptor"
       class="org.springframework.transaction.interceptor.Transacti
    <property name="transactionManager" ref="transactionManager" />
    <property name="transactionAttributes">
       <props>
        <prop key="save">PROPAGATION_REQUIRED</prop>
       </props>
    </property>
    </bean>

    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransact
      <property name="dataSource" ref="dataSource" />
      <property name="sessionFactory" ref="sessionFactory" />
    </bean>

</beans>
```

## 事务属性

在事务拦截器，必须定义的事务的属性“传播行为”应使用。这意味着，如果一个事务“ProductBoImpl.save()方法被调用另外的”productQohBo.save()'方法，事务应该怎么传播？它能继续在现有的事务中运行？或者为自己开始一个新的事务。

由Spring支持传播的 7种类型：

- PROPAGATION_REQUIRED – 支持当前事务;如果不存在则创建一个新的。
- PROPAGATION_SUPPORTS – 支持当前事务;如果不存在执行非事务。
- PROPAGATION_MANDATORY – 支持当前事务;如果当前不存在事务抛出异常。
- PROPAGATION_REQUIRES_NEW – 创建一个新的事务，如果当前事务暂停。
- PROPAGATION_NOT_SUPPORTED – 不支持当前的事务;而始终执行非事务。
- PROPAGATION_NEVER – 不支持当前的事务;如果当前事务存在则抛出异常。
- PROPAGATION_NESTED – 如果当前存在事务嵌套事务中执行，表现与 PROPAGATION_REQUIRED 一样。

在大多数情况下，可能只需要使用PROPAGATION_REQUIRED。此外，必须定义方法来支持这个事务属性。方法名支持通配符格式，save*会匹配所有的方法名 以save(...)开始的方法 。

## 事务管理器

在Hibernate事务，需要使用 HibernateTransactionManager 。 如果只对付纯JDBC，useDataSourceTransactionManager; 而如果是 JTA，需要使用JtaTransactionManager 。

# 4.代理工厂bean

创建一个新的代理工厂bean的ProductBo，并设置“interceptorNames”属性。

```
<!-- Product business object -->
   <bean id="productBo" class="com.yiibai.product.bo.impl.ProductBo
       <property name="productDao" ref="productDao" />
       <property name="productQohBo" ref="productQohBo" />
   </bean>

   <!-- Product Data Access Object -->
   <bean id="productDao" class="com.yiibai.product.dao.impl.Product
       <property name="sessionFactory" ref="sessionFactory"></prope
   </bean>

   <bean id="productBoProxy"
    class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target" ref="productBo" />
    <property name="interceptorNames">
        <list>
            <value>transactionInterceptor</value>
        </list>
    </property>
  </bean>
```

运行它

```
Product product = new Product();
    product.setProductCode("ABC");
    product.setProductDesc("This is product ABC");

    ProductBo productBo = (ProductBo)appContext.getBean("productBoP
    productBo.save(product, 100);
```

代理 bean' productBoProxy'和 save()方法是支持事务，现在，里面 productBo.save()方法任何异常会导致整个事务回滚，没有数据会被插入到数据库中。下载代码 – http://pan.baidu.com/s/1qXynbo4

# 参考


1. http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/transaction/TransactionDefinition.html
2. http://static.springsource.org/spring/docs/2.5.x/reference/transaction.html

# Spring在bean配置文件中定义电子邮件模板 - Spring教程

在上一篇Spring电子邮件教程，硬编码的所有电子邮件属性和消息的方法体中的内容，这是不实际的，应予以避免。应该考虑在Spring bean 配置文件中定义电子邮件模板。

## 1.Spring的邮件发件人

Java类使用 Spring的MailSender接口发送电子邮件，并使用 String.Format 传递变量bean配置文件替换电子邮件中的 '%s'。

*File : MailMail.java*

```
package com.yiibai.common;

import org.springframework.mail.MailSender;
import org.springframework.mail.SimpleMailMessage;

public class MailMail
{
    private MailSender mailSender;
    private SimpleMailMessage simpleMailMessage;

    public void setSimpleMailMessage(SimpleMailMessage simpleMailMe
        this.simpleMailMessage = simpleMailMessage;
    }

    public void setMailSender(MailSender mailSender) {
        this.mailSender = mailSender;
    }

    public void sendMail(String dear, String content) {

       SimpleMailMessage message = new SimpleMailMessage(simpleMail

       message.setText(String.format(
            simpleMailMessage.getText(), dear, content));

       mailSender.send(message);

    }
}
```

# 2. Bean的配置文件

定义电子邮件模板“customeMailMessage' 和邮件发件人信息的bean配置文件。

*File : Spring-Mail.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="mailSender" class="org.springframework.mail.javamail.Java
    <property name="host" value="smtp.gmail.com" />
    <property name="port" value="587" />
    <property name="username" value="username" />
    <property name="password" value="password" />

    <property name="javaMailProperties">
         <props>
               <prop key="mail.smtp.auth">true</prop>
               <prop key="mail.smtp.starttls.enable">true</prop>
                </props>
    </property>
</bean>

<bean id="mailMail" class="com.yiibai.common.MailMail">
    <property name="mailSender" ref="mailSender" />
    <property name="simpleMailMessage" ref="customeMailMessage" />
</bean>

<bean id="customeMailMessage"
    class="org.springframework.mail.SimpleMailMessage">

    <property name="from" value="from@no-spam.com" />
    <property name="to" value="to@no-spam.com" />
    <property name="subject" value="Testing Subject" />
    <property name="text">
       <value>
        <![CDATA[
            Dear %s,
            Mail Content : %s
        ]]>
       </value>
        </property>
</bean>

</beans>
```

# 4. 运行它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
           new ClassPathXmlApplicationContext("applicationContext.x

        MailMail mm = (MailMail) context.getBean("mailMail");
        mm.sendMail("Yiibai", "This is text content");

    }
}
```

输出

```
Dear Yiibai,
 Mail Content : This is text content
```

代码下载 – http://pan.baidu.com/s/1c0UPsFA

# Spring发送带附件邮件 - Spring教程

下面是一个例子使用Spring通过Gmail SMTP服务器来发送电子邮件附件。为了包含附件的电子邮件，你必须使用 Spring的JavaMailSender及MimeMessage 来代替MailSender＆SimpleMailMessage。

## 2.Spring的邮件发件人

必须使用 JavaMailSender 代替 MailSender 发送附件，并用 MimeMessageHelper 附加的资源。在这个例子中，它会得到 “c:\log.txt” 从文件系统(FileSystemResource)作为电子邮件附件的文本文件。

除了文件系统，您还可以从URL路径(UrlResource对象)，类路径(使用ClassPathResource)，InputStream(InputStreamResource)的任何资源......请参考Spring 的 AbstractResource 类的实现。

*File : MailMail.java*

```
package com.yiibai.common;

import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;

import org.springframework.core.io.FileSystemResource;
import org.springframework.mail.MailParseException;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;

public class MailMail
{
    private JavaMailSender mailSender;
    private SimpleMailMessage simpleMailMessage;

    public void setSimpleMailMessage(SimpleMailMessage simpleMailMe
        this.simpleMailMessage = simpleMailMessage;
    }

    public void setMailSender(JavaMailSender mailSender) {
        this.mailSender = mailSender;
    }

    public void sendMail(String dear, String content) {

       MimeMessage message = mailSender.createMimeMessage();

       try{
        MimeMessageHelper helper = new MimeMessageHelper(message, t

        helper.setFrom(simpleMailMessage.getFrom());
        helper.setTo(simpleMailMessage.getTo());
        helper.setSubject(simpleMailMessage.getSubject());
        helper.setText(String.format(
            simpleMailMessage.getText(), dear, content));

        FileSystemResource file = new FileSystemResource("C:\\log.t
        helper.addAttachment(file.getFilename(), file);

         }catch (MessagingException e) {
        throw new MailParseException(e);
         }
         mailSender.send(message);
         }
}
```

# 3. Bean配置文件

配置 mailSender bean，电子邮件模板，并指定Gmail的SMTP服务器电子邮件的详细信息。

*File : Spring-Mail.xml*

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="mailSender" class="org.springframework.mail.javamail.Java
    <property name="host" value="smtp.gmail.com" />
    <property name="port" value="587" />
    <property name="username" value="yiibai.com@gmail.com" />
    <property name="password" value="password" />

    <property name="javaMailProperties">
        <props>
               <prop key="mail.smtp.auth">true</prop>
               <prop key="mail.smtp.starttls.enable">true</prop>
           </props>
    </property>
</bean>

<bean id="mailMail" class="com.yiibai.common.MailMail">
    <property name="mailSender" ref="mailSender" />
    <property name="simpleMailMessage" ref="customeMailMessage" />
</bean>

<bean id="customeMailMessage"
    class="org.springframework.mail.SimpleMailMessage">

    <property name="from" value="from@no-spam.com" />
    <property name="to" value="to@no-spam.com" />
    <property name="subject" value="Testing Subject" />
    <property name="text">
    <value>
        <![CDATA[
            Dear %s,
            Mail Content : %s
        ]]>
    </value>
    </property>
</bean>

</beans>
```

## 4. 运行它

```
package com.yiibai.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.

        MailMail mm = (MailMail) context.getBean("mailMail");
        mm.sendMail("Yiibai", "This is text content");

    }
}
```

输出结果

```
Dear Yiibai,
 Mail Content : This is text content
 Attachment : log.txt
```

下载代码 – http://pan.baidu.com/s/1jHn9VLW

# Spring依赖注入servlet会话监听器 - Spring教程

Spring提供了一个 “ContextLoaderListener” 监听器，以使 Spring 依赖注入到会话监听器。 在本教程中，通过添加一个 Spring 依赖注入一个bean 到会话监听器修改 HttpSessionListener 例子。

## 1. Spring Beans

创建一个简单的计数服务来打印创建的会话总数。

*File : CounterService.java*

```
package com.yiibai.common;

public class CounterService{

    public void printCounter(int count){
        System.out.println("Total session created : " + count);
    }

}
```

*File : counter.xml* – Bean配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="counterService" class="com.yiibai.common.CounterServi

</beans>
```

## 2. WebApplicationContextUtils

使用“WebApplicationContextUtils”来获得 Spring 上下文，以后可以得到任何声明 Spring的Bean 在一个正常的 Spring 方式。

*File : SessionCounterListener.java*

```
package com.yiibai.common;

import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
import org.springframework.context.ApplicationContext;
import org.springframework.web.context.support.WebApplicationContex

public class SessionCounterListener implements HttpSessionListener

     private static int totalActiveSessions;

     public static int getTotalActiveSession(){
           return totalActiveSessions;
     }

    @Override
    public void sessionCreated(HttpSessionEvent arg0) {
           totalActiveSessions++;
           System.out.println("sessionCreated - add one session int
           printCounter(arg0);
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent arg0) {
           totalActiveSessions--;
           System.out.println("sessionDestroyed - deduct one sessio
           printCounter(arg0);
    }

    private void printCounter(HttpSessionEvent sessionEvent){

          HttpSession session = sessionEvent.getSession();

          ApplicationContext ctx =
                WebApplicationContextUtils.
                      getWebApplicationContext(session.getServletCo

          CounterService counterService =
                      (CounterService) ctx.getBean("counterService"

          counterService.printCounter(totalActiveSessions);
    }
}
```

# 3. 集成

唯一的问题是，如何使 Web 应用程序知道在哪里可以加载 Spring bean 配置文件？秘诀是在“web.xml”文件中。

1. 注册“ContextLoaderListener”作为第一个监听器，使Web应用程序知道Spring上下文加载程序。
2. 配置“contextConfigLocation”，并定义Spring的bean配置文件。

*File : web.xml*

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/Spring/counter.xml</param-value>
  </context-param>

  <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
  </listener>

  <listener>
    <listener-class>
            com.yiibai.common.SessionCounterListener
        </listener-class>
  </listener>

  <servlet>
    <servlet-name>Spring DI Servlet Listener</servlet-name>
    <servlet-class>com.yiibai.common.App</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Spring DI Servlet Listener</servlet-name>
    <url-pattern>/Demo</url-pattern>
  </servlet-mapping>

</web-app>
```

*File : App.java*

```
package com.yiibai.common;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class App extends HttpServlet{

  public void doGet(HttpServletRequest request, HttpServletResponse
        throws IOException{

        HttpSession session = request.getSession(); //sessionCreate
        session.setAttribute("url", "yiibai.com");
        session.invalidate();  //sessionDestroyed() is executed

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Spring Dependency Injection into Servlet L
        out.println("</body>");
        out.println("</html>");

   }
}
```

启动Tomcat，并访问 URL “http://localhost:8080/7.2-SpringDemo/”

输出结果

```
sessionCreated - add one session into counter
Total session created : 1
sessionDestroyed - deduct one session from counter
Total session created : 0
```

看看控制台的输出，会得到通过 Spring DI 记数服务的 bean，并打印会话的总数。

## 总结

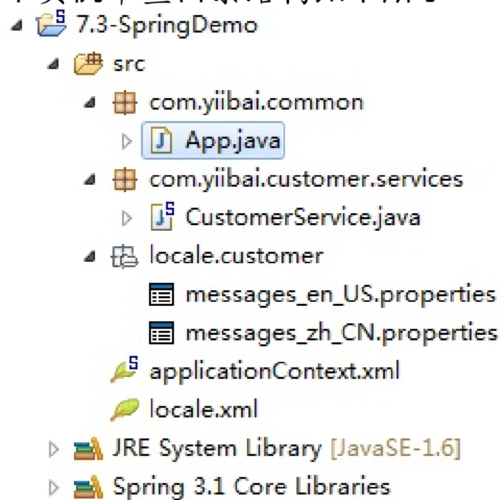
在Spring中，“ContextLoaderListener”是一个通用的方法集成Spring依赖注入到几乎所有的Web应用程序。下载代码 – http://pan.baidu.com/s/1bpmhQY

# Spring资源捆绑ResourceBundleMessageSource示例 - Spring教程

在Spring中，你可以使用ResourceBundleMessageSource从属性文件，基于解决所选择的区域设置文本信息。参见下面的例子：

## 1. 目录结构

本实例审查目录结构如下所示：




## 2. Properties文件

创建两个属性文件，一个是英文字符(messages_en_US.properties)，另一种为中国字符(messages_zh_CN.properties)。把它放到项目的类路径中(见上图)。

*File : messages_en_US.properties*

```
customer.name=Yong Mook Kim, age : {0}, URL : {1}
```

*File : messages_zh_CN.properties*

```
customer.name=\u6613\u767E\u6559\u7A0B, age \: {0}, URL \: {1}
```

在'\u6613\u767E\u6559\u7A0B' 是中文 Unicode 字符。

## 3. Bean配置文件

包含属性文件到bean的配置文件。这两个“messages_en_US.properties”和“messages_zh_CN.properties”考虑放 Spring 的一个文件中， 只需要包含文件名一次，Spring 会自动找到正确的语言环境。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

    <bean id="messageSource"
        class="org.springframework.context.support.ResourceBundleMe
        <property name="basename">
            <value>locale\customer\messages</value>
        </property>
    </bean>

</beans>
```

P.S 假设这两个文件位于“项目根目录”文件夹中。

## 4. 运行它

```
package com.yiibai.common;

import java.util.Locale;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC

public class App {
    public static void main(String[] args) {

        ApplicationContext context
            = new ClassPathXmlApplicationContext("locale.xml");

        String name = context.getMessage("customer.name",
                new Object[] { 28,"http://www.yiibai.com" }, Locale

        System.out.println("Customer name (English) : " + name);

        String namechinese = context.getMessage("customer.name",
                new Object[] {29, "http://www.yiibai.com" },
                                        Locale.SIMPLIFIED_CHINESE);

        System.out.println("Customer name (Chinese) : " + namechine

    }
}
```

输出结果

```
Console  Problems  Javadoc  Declaration
<terminated> App (41) [Java Application] D:\Program Files (x86)\MyEclipse\Common\binary\com.sun.java.jdk.win32.x86
log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Customer name (English) : Yiibai, age : 28, URL : http://www.yiibai.com
Customer name (Chinese) : 易百教程, age : 28, URL : http://www.yiibai.com
Customer name (English) : Yiibai, age : 28, URL : http://www.yiibai.com
Customer name (Chinese) : 易百教程, age : 29, URL : http://www.yiibai.com
```

## 解释：

1. 在 context.getMessage()，第二个参数是消息参数，必须把它作为对象数组。如果没有参数值可用，可以只传递一个空(null)。

```
context.getMessage("customer.name",null, Locale.US);
```

1. Locale.US将从“messages_en_US.properties”获取信息，而 Locale.SIMPLIFIED_CHINESE将从“messages_zh_CN.properties”获取信息。

下载代码 – http://pan.baidu.com/s/1gdPt5Ur

# Struts2教程

Apache Struts 2 是一种流行的 Java模型 - 视图 - 控制器（MVC）框架，成功地结合了 WebWork和Struts1.x 两种 web 框架。

Apache Struts2与Struts1完全不同，它的核心功能都是使用拦截实现“值栈”的概念，OGNL表达式和Struts2标签来解决应用程序数据，很多注解和约定，使这个框架更加易于使用。 在本教程中，它提供了许多使用Struts2的MVC框架的实例和解释。

欢迎来到易百教程学习Struts2。

## Struts2 快速入门

Struts2快速入门的例子。

- Struts 2 hello world (XML版本) 使用XML文件的Struts 2 Hello World 示例
- Struts 2 Hello World (注解版本) 使用注释的Struts 2 Hello World 示例
- @ResultPath 注释示例 @ResultPath 注释说明和示例

## Struts2 配置

任何关于 Struts2 的配置文件。

- 多个Struts配置文件示例 拆分大的Struts配置文件分成多个小的配置文件。
- Struts2 命名空间配置和解释 Struts2命名空间是一个新的概念，用来处理多个模块，由下式给出一个命名空间对应的每个模块。
- Struts2开发模式 启用了Struts2开发模式将会使调试变得更容易。
- 如何删除Struts2动作的后缀扩展名 默认的“.action”扩展名是丑陋的，并不是用户友好的，可以删除或用另一个扩展名来取代它。

## Struts2 动作和表单

Struts 2的动作和表单数据管理。

- 使用Struts2动作Struts 2的动作说明和示例。
- Struts 2 ActionError & ActionMessage Example Struts 2的ActionError和ActionMessage的解释和例子。
- Struts 2 ModelDriven example 自动将的表单数据传输到对象。

## Struts2 拦截器

关于Struts 2的拦截器。

- 映射拦截动作配置拦截器动作。
- 重写拦截器参数几种方法来覆盖拦截器的参数。
- 拦截器栈的例子拦截器堆栈用于建立一组的拦截器，以再利用。
- 创建自己的拦截器创建自己的拦截器指南，以满足您的需求。
- execAndWait拦截器例子一个非常方便的拦截器长时间运行动作在后台，显示用户的自定义的等待页面。

## Struts 2 UI标签

Struts 2的UI标签，来渲染HTML表单和非表单组件。

- TextBox文档示例 Struts2 <s:textfield> 文本实例.
- Password 示例 Struts2 <s:password> 密码实例.
- Hidden隐藏值示例 Struts2 <s:hidden> 隐藏值例子
- Textarea - 文本域 Struts2 <s:textarea> textarea例子
- Radio 单选按键示例 Struts2 <s:radio> radio单选按钮例子
- 预选单选按钮 预选单选按钮值示例
- 复选框heckbox 示例 Struts2 <s:checkbox> 复选框示例
- 复选框checkboxes 示例 Struts2 <s:checkboxlist> 多发复选框的例子
- 设置复选框的默认值 设置多个复选框的默认值
- 下拉框示例 Struts2 <s:select> 下拉框例子
- 自动选择下拉框示例 自动选择下拉框值指南
- 组合框示例 Struts2 <s:combobox> 组合框的例子
- head 示例 Struts2 <s:head>, 呈现一个HTML头组件
- 文件上传示例 Struts2 <s:file> 文件上传示例
- 多文件上传示例 Struts2 <s:file> 多文件上传示例
- 级联选择示例 Struts2 <s:doubleselect>, 创建两个HTML下拉框，当第一下拉列表中选择，第二下拉列表将相应地改变
- updownselect 示例 Struts2 <s:updownselect>, 创建一个带有按钮，向上或向下移动在选择组件的选项选择HTML组件。
- optiontransferselect 示例 Struts2 <s:optiontransferselect>, 两个“updownselect”选择组件排列在左侧和右侧，在它们中间包含有按钮来移动自己的选择选项。
- datetimepicker 日期选择 Struts2 <s:datetimepicker>, 将呈现一个文本框和追加后面的日历图标，单击日历图标上会提示的日期时间选择器组件。
- autocompleter自动完成示例 Struts2 <s:autocompleter>, 一个组合框，会自动提示下拉的提示菜单，在用户输入文本框时。
- autocompleter + JSON 示例 举个例子，使用JSON数据填充到autocompleter组件。

## Struts 2 控制标签

在 Struts2 中的控制标签或逻辑标签，用来做条件处理，迭代，处理和显示数据。

- <s:iterator>标签迭代示例 Struts2迭代器标签用来迭代一个值，它可以是任何的 java.util.Collection 或 java.util.Iterator

- <s:if>, <s:elseIf>, <s:else> 标签示例 Struts2 if,elseif和else标签被用来执行基本条件检查。
- <s:append>标签示例 Struts2 <s:append>标签用来组合几个迭代器（由列表或映射创建）到一个迭代器
- <s:generator> 标签示例 struts2 <s:generator>标记用于基于在页中提供“val”属性，以产生一个迭代。
- <s:merge>标签示例 Struts2 <s:merge>标签用来合并几个迭代器（以列表或映射创建）成一个迭代器。
- <s:sort>标签示例 Struts2 <s:sort>是用于排序一个列表，它通过使用java.util.Comparator 来实现。
- subset tag example Struts2 <s:subset>标记用于输出一个迭代元素的子集或部分。

## Struts2 数据标签

Struts2数据标签，从ValueStack中获取数据，或将数据放入ValusStack。

- <s:a>标签示例 Struts2的<s:a>标签被用于渲染HTML的“<a>”标签。
- <s:action>标签示例 Struts2的<s:action>标签用来直接在一个JSP页面中调用Action类
- <s:bean>标签示例 Struts2的<s:bean>标签用来在JSP页面中实例化一个类
- <s:date>标签示例 Struts2的<s:date>标签用来在JSP页面格式Date对象。
- <s:debug>标签示例 Struts 2的<s:debug>标签是一个非常有用的调试标记，用于输出“值栈”的内容，并在JSP页面中输出“堆栈上下文”的详细信息。
- <s:include>标签示例 Struts 2的<s:include>标签用来直接包含JSP或HTML页面到当前页面。
- <s:i18n>标签示例 Struts 2的<s:i18n>标签用来获取声明的资源包，而不仅仅是资源包，也可获取当前操作相关联的消息。
- <s:param>标签示例 Struts2的<s:param>标签用来参数化其他标签。
- <s:property>标签示例 Struts2的<s:property>标签用来从一个类获取当前默认Action类的属性值。
- <s:push>标签示例 Struts2的<s:push>标签用来推值到堆栈的顶部，以便它可以容易访问或参考。
- <s:set>标签示例 Struts2的<s:set>标签用来在指定的范围内（应用，会话，请求， 页面，或动作）赋值给一个变量
- <s:text>标签示例 Struts2的<s:text>标签用于从操作类取出资源包消息
- <s:url>标签示例 Struts2的 <s:url> 标签用来创建一个URL，并输出作为文本格式

## Struts2 资源包和本地化

Struts2的资源包来支持网络定位功能（多语言）

- 资源包使用示例 Struts2的资源包的解释和例子
- i18n 或本地化示例 一个Struts 2的国际化和多语言的例子来说明如何使用资源包来显示不同语言的消息

- key 属性示例 Struts 2 key属性在UI组件是处理本地化的常用方法，也UI标签编码的一个非常有效的方法
- Chinese 本地化问题 一个常见的中国本地化的问题
- 配置全局资源包 配置Struts2的全局资源包指南

## Struts2主题

Struts2 的布局是由“XHTML”主题设计的，所以了解 Struts2 主题概念是必须的。

- Struts2 主题和模板 Struts 2主题和模板的说明和示例

## Struts2集成其它框架

Struts2与任何他人框架的整合 - Spring, Hibernate, Quartz, Log4j...

- Struts2 + Spring集成实例 Struts2和Spring框架集成。
- Struts2 + Quartz调度集成实例 Struts2和Quartz调度框架集成。
- Struts2 + Hibernate 集成实例 集成 Struts2 和 Hibernate 框架。
- Struts2 + Hibernate使用“Full Hibernate Plugin"插件集成 使用 “Full Hibernate Plugin”集成Struts2 和Hibernate3
- Struts2 + Spring + Hibernate集成实例 集成Struts2，Spring和Hibernate三个框架。
- Struts 2 + Log4j 集成实例 集成Struts 2 和Log4j 框架.

## Struts2 FAQ

- FilterDispatcher 和 StrutsPrepareAndExecuteFilter区别？ 关于开发问 filterdispatcher 和 strutsprepareandexecutefilter 之间的差异。
- 在Struts2中获取 HttpServletRequest 获取 Struts2 HttpServletRequest对象的实例。
- 在Struts2获取HttpServletResponse对象 在Struts2中如何获取 HttpServletResponse 对象实例
- 在Struts2中如何获取ServletContext对象 在Struts2获取ServletContext对象实例
- 在Struts2中配置静态参数(有示例代码) 在Struts2配置静态参数实例
- Struts2下载文件实例（有实例代码） Struts2实现下载文件实例
- Struts2 和 JSON 实例（有实例代码） 举个例子来说明Struts2集成JSON数据。

## Struts2 参考

- Struts2 官方文档
- Struts2 通用标签
- http://en.wikipedia.org/wiki/Apache_Struts

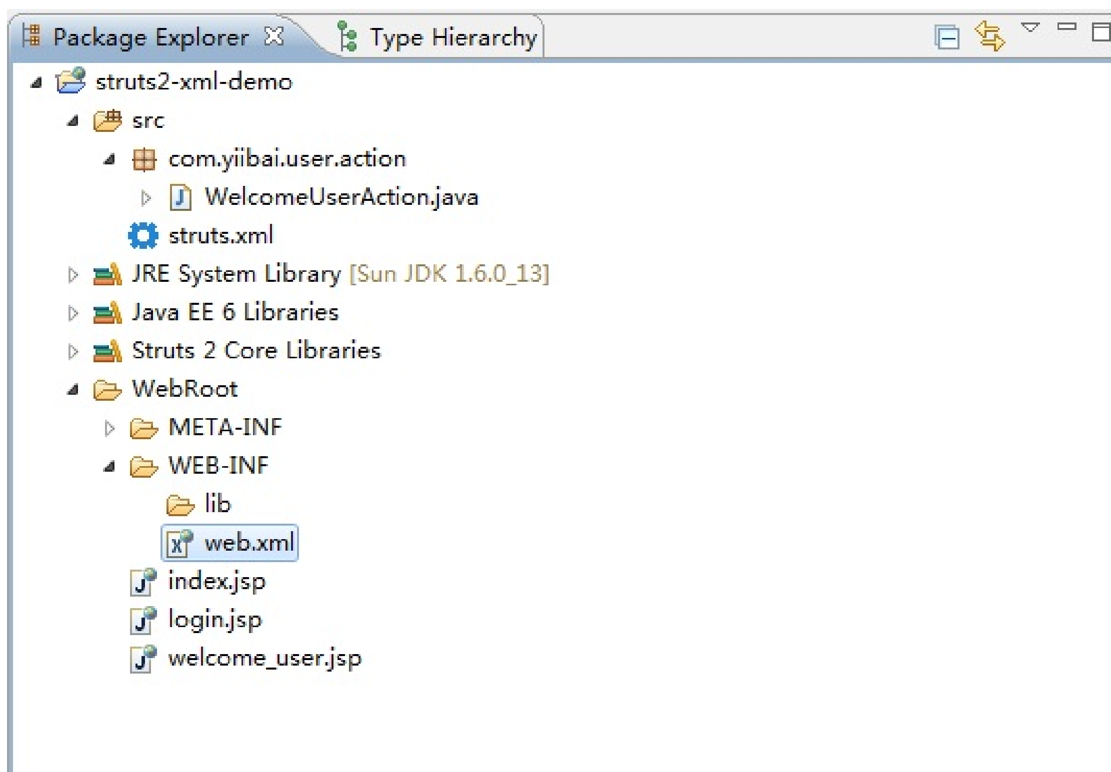
# Struts 2 hello world (XML版本) - Struts2教程

在这个例子中，我们将学习如何在Struts 2中创建一个Hello World例子。

使用以下库或工具：

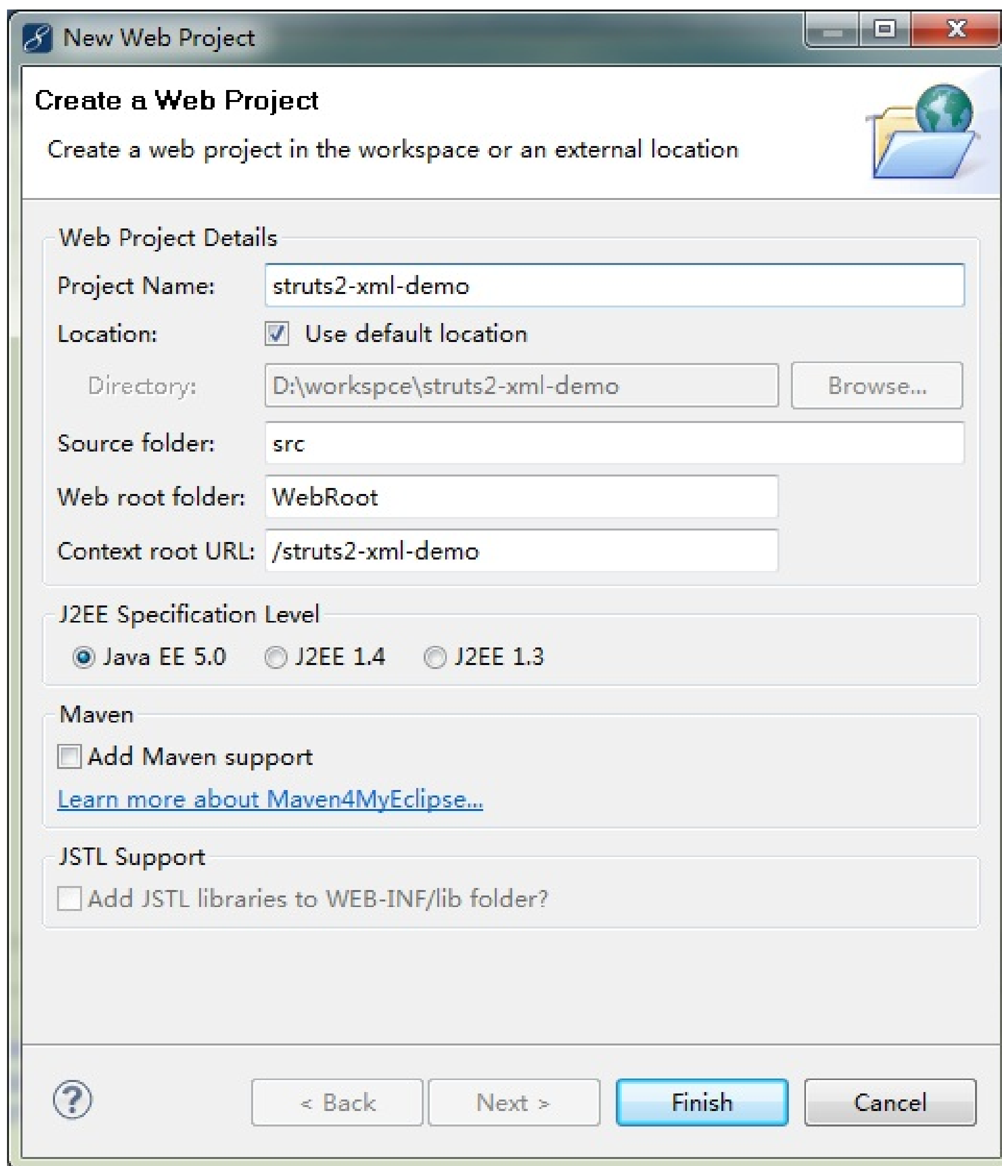
- MyEclipse 10
- Struts 2.1

整个工程结构如下图所示：

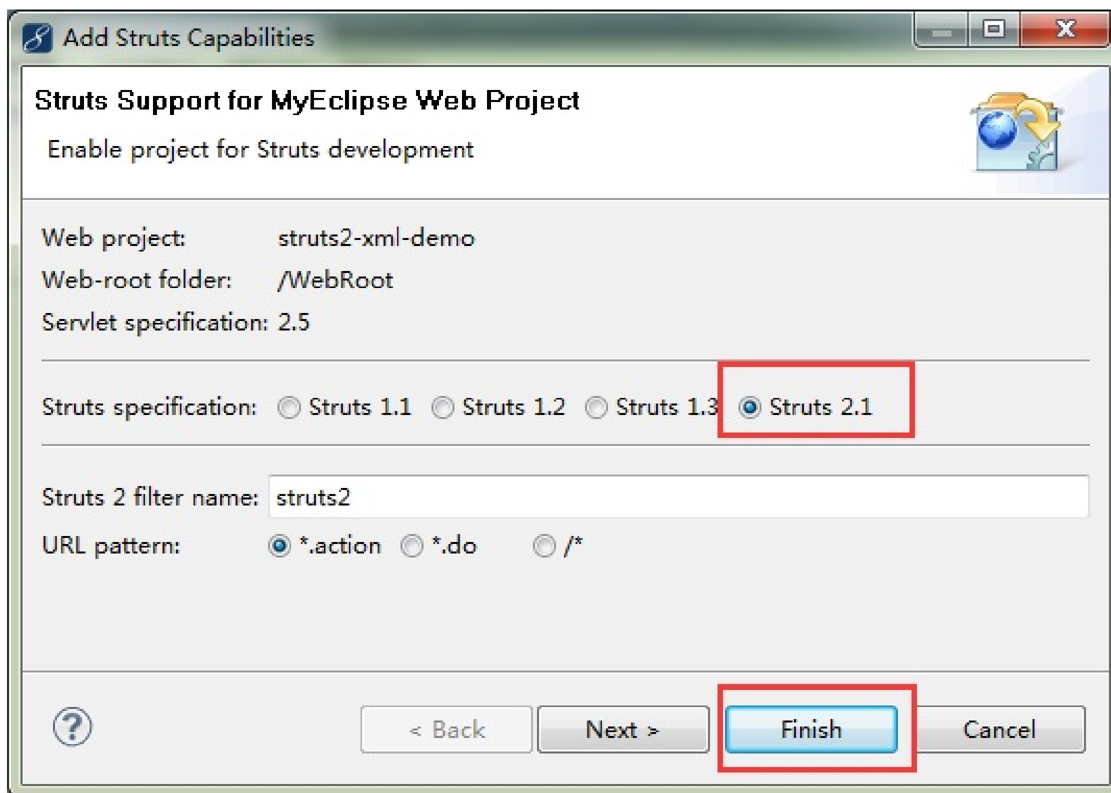



## 1. 创建一个Web项目工程

启动打开 MyEclipse，创建一个Web工程名称为：struts2-xml-demo，选择 File -> New -> Web Project， 如下图所示：




在这个项目上添加 struts2 的支持，右键点击 struts2-xml-demo 工程，选择 MyEclipse -> Add Struts Capabilities，在弹出的对话框中选择 Strut 2.1，如下图所示：




## 2. JSP视图文件

这是一个JSP登录页面，它使用Struts2标签来显示用户名，密码输入框和提交按钮。

Fie : login.jsp

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head></head>
<body>
    <h1>Struts 2 Hello World Example</h1>

    <s:form action="Welcome">
        <s:textfield name="username" label="Username" />
        <s:password name="password" label="Password" />
        <s:submit />
    </s:form>

</body>
</html>
```

文件: *welcome_user.jsp* – 一个JSP视图用来页面显示欢迎信息给用户。

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head></head>
<body>
    <h1>Struts 2 Hello World 示例</h1>

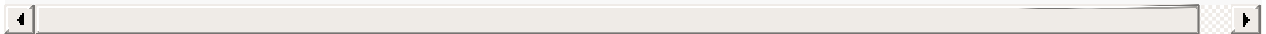
    <h2>
        Hello
        <s:property value="username" />
    </h2>

</body>
</html>
```

对 Struts1 和 Struts2 有非常相似的UI标签语法，只是在命名HTML元素，例如，术语有一点不同：

**Struts 1**

```
<%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<html:form action="Welcome">
   <html:text property="username"/>
</html:form>
```



**Struts 2**

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<s:form action="Welcome">
    <s:textfield name="username" label="Username"/>
</s:form>
```

# 5. 动作，所有的业务逻辑放在这里

一个简单的 Struts2 的 Action 类，它里面声明的所有业务逻辑。

File : WelcomeUserAction.java

```
package com.yiibai.user.action;

/**
 *
 * @author yiibai.com
 *
 */
public class WelcomeUserAction {

    private String username;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    // all struts logic here
    public String execute() {

        return "SUCCESS";

    }
}
```

在Struts2中，Action类实现任何接口或扩展任何类不是必需的，但它需要创建一个execute()方法来实现所有的业务逻辑，并返回一个字符串值，告诉用户重定向到哪里。

注意 您可能会看到一些用户实现 com.opensymphony.xwork2.Action 类， 但它是完全可选的(不是必须的)，因为com.opensymphony.xwork2.Action只是提供一些方便的常量。Struts1中的Action类需要扩展org.apache.struts.action.Action。 但是，Struts 2的Action类是可选的，但是仍然允许执行com.opensymphony.xwork2.Action的一些方便的常量，或者扩展com.opensymphony.xwork2.ActionSupport 对于一些常见的默认动作执行的功能。

# 5. Struts配置文件

Strut配置文件是用来连接所有的东西在一起。 XML文件名必须是 “struts.xml”。在这个实例中，它位于

*File : struts.xml*

```
_<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
<struts>
    <package name="user" namespace="/User" extends="struts-default'
        <action name="Login">
            <result>/login.jsp</result>
        </action>
        <action name="Welcome" class="com.yiibai.user.action.Welcon
            <result name="SUCCESS">/welcome_user.jsp</result>
        </action>
    </package>
</struts>_
```

声明包和包含动作类，动作类是不言自明的，但你仍可能会感兴趣下面的新标签：

**1. package name="user"** 就在包名，并不真正去关心它。

**2. namespace="/User"** 它用于匹配"/User"URL模式。

注意 实际上，Struts2的命名空间相当于Struts的1多个功能模块

**3. extends="struts-default"** 这意味着该包是扩展了struts-default 包组件和拦截器，这是在struts-default.xml中文件中声明的，位于struts2-core.jar 文件的根目录。

# 6. web.xml

配置Web应用程序部署描述符(web.xml)文件Struts2的集成到Web项目。

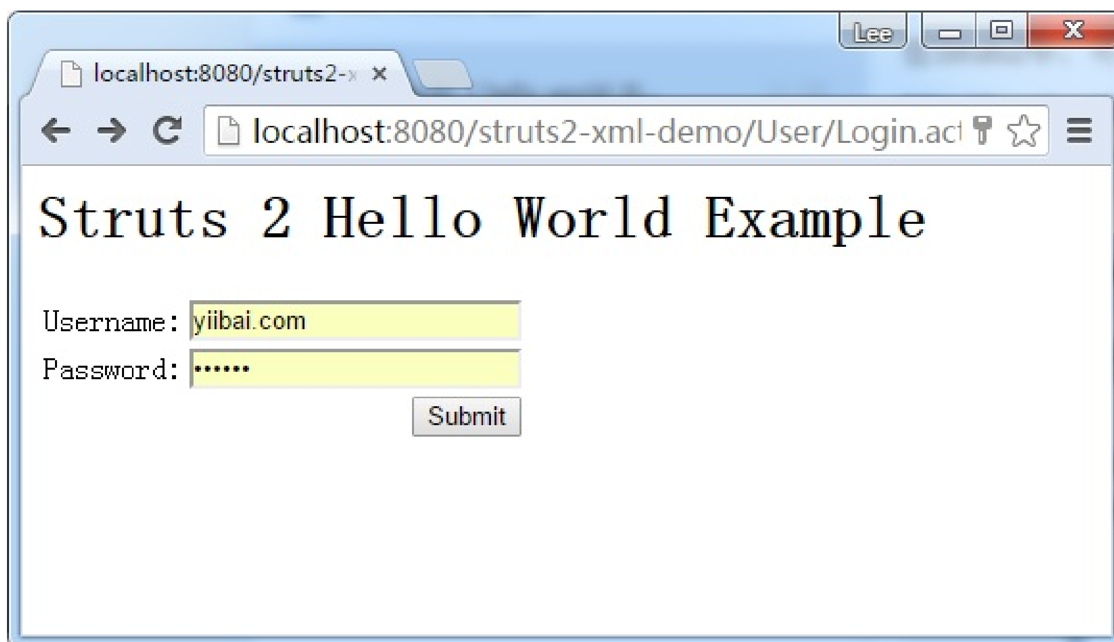
*File web.xml*

```
_<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <display-name></display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <filter>
      <filter-name>struts2</filter-name>
      <filter-class>
          org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndE
      </filter-class>
  </filter>
  <filter-mapping>
      <filter-name>struts2</filter-name>
      <url-pattern>*.action</url-pattern>
  </filter-mapping></web-app>_
```

## 7. 运行测试结果

在Struts2中，可以直接使用.action后缀访问操作类。如下URL：

http://localhost:8080/struts2-xml-demo/User/Login.action




提交后到 http://localhost:8080/Struts2Example/User/Welcome.action 显示如下：


localhost:8080/struts2-x
localhost:8080/struts2-xml-demo/User/Welcome.ac
Struts 2 Hello World 示例
Hello yiibai.com

# Struts2注解示例 - Struts2教程

在这个教程，我们重复使用以前 STRUST2 Hello World(XML版本)的例子，并将其转换成注解版本。

## Struts2 注解概念

Struts2注解是由Struts 2的约定插件的支持，所以，必须要了解其背后的“扫描方法”和“命名转换”机制的魔力。

## 1. 扫描方法

许多Struts 2的文章或书籍说，可以配置过滤器的“init-param”或“struts.convention.action.packages”告诉Struts2，其中扫描注解的类。 例如，

**web.xml**

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</fi
  <init-param>
    <param-name>actionPackages</param-name>
    <param-value>com.yiibai.common</param-value>
  </init-param>
</filter>
```

从测试(Struts22.1.6和2.1.8版本)，这是不正确的，不管你把在“param-value”还是“struts.convention.action.packages“, 在Struts 2会忽略它，并只扫描指定的文件夹命名：struts, struts2, action 或 actions 。

下面是扫描工作

1. 扫描其位于包的命名注解的类 “struts, struts2, action 或 actions“.
2. 接着，扫描相匹配下列任一条件的文件：
   - 实例了 com.opensymphony.xwork2.Action 接口。
   - 扩展了 com.opensymphony.xwork2.ActionSupport 类
   - 文件名用动作(例如：UserAction，LoginAction)结束

详细请查看这里Struts 2 约定插件文件

## 2. 命名转换器

Struts 2的约定插件将所有的注解操作文件名转换为指定的格式。

例如 : LoginAction.java

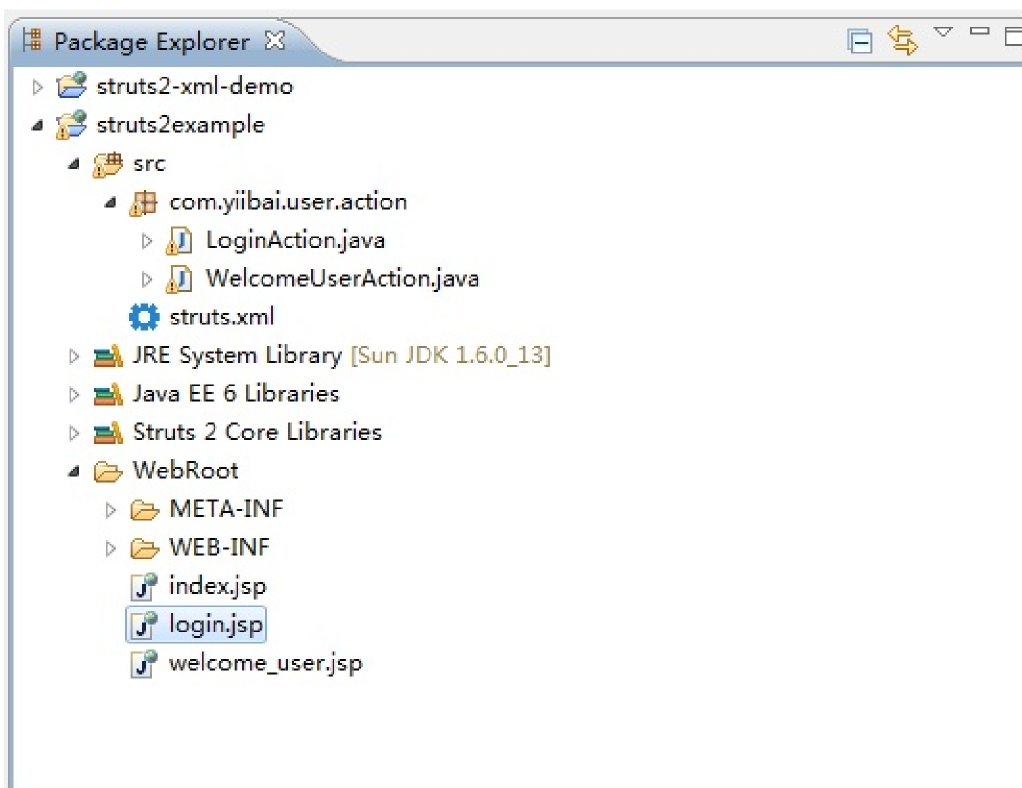
1. 首先，去掉“Action”字符在文件名的末尾，如果存在的话。
2. 其次，转换文件名的第一个字母为小写。

因此，去除结束并转换第一个字母为小写后，LoginAction.action 将变为 login.action。Struts2约定插件的“扫描方法”和“命名转换”特性真正带来了很多的便利和好处，只有当你的Struts2项目正确下面的命名约定才会带来好处; 否则，这将是一场灾难。

# Struts 2 注解例子

现在是时候开始转换过程了，我们使用MyEclipse 10 创建一个工程为 : struts2example。

最终的项目结构




# 2. LoginAction

扩展ActionSupport并创建了LoginAction，什么也不做，ActionSupport 默认返回 “success” 字符串,这将匹配 @Result 并重定位到 “pages/login.jsp“.

注解版本

```
package com.yiibai.user.action;

import org.apache.struts2.convention.annotation.Namespace;
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.ResultPath;

import com.opensymphony.xwork2.ActionSupport;

@Namespace("/User")
@ResultPath(value="/")
@Result(name="success",location="/login.jsp")
public class LoginAction extends ActionSupport{

}
```

**XML** 实现版本

```
**<package name="user" namespace="/User" extends="struts-default">
    <action name="Login">
        <result>/login.jsp</result>
    </action>
</package>**
```

# 3. WelcomeUserAction

重写execute()方法并指定 @Action 和 @Result 注解。

注解版本

```
package com.yiibai.user.action;

import org.apache.struts2.convention.annotation.Action;
import org.apache.struts2.convention.annotation.Namespace;
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.ResultPath;

import com.opensymphony.xwork2.ActionSupport;

@Namespace("/User")
@ResultPath(value="/")
public class WelcomeUserAction extends ActionSupport{

    private String username;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @Action(value="Welcome", results={
        @Result(name="success",location="welcome_user.jsp")
    })
    public String execute() {

        return SUCCESS;

    }
}
```

**XML 实现版本**

```
<package name="user" namespace="/User" extends="struts-default">
   <action name="Welcome" class="com.yiibai.user.action.WelcomeUse
    <result name="SUCCESS">/welcome_user.jsp</result>
   </action>
</package>
```

Struts 2 注解 – @Action, @Result 和 @Namespace 不言自明，可以将它与XML比较。@ResultPath 可能需要一点点的解释，请参阅本 @ResultPath示例

# 4. JSP视图页面

普通JSP视图页面来接受用户名和密码后点击提交按钮，并重定向到一个欢迎页面。

**login.jsp**

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head></head>
<body>
    <h1>Struts 2 注解示例</h1>

    <s:form action="Welcome">
        <s:textfield name="username" label="用户名" />
        <s:password name="password" label="密码" />
        <s:submit value="提交"/>
    </s:form>

</body>
</html>
```

**welcome_user.jsp**

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head></head>
<body>
<h1>Struts 2 注解示例</h1>

<h4>您好, <s:property value="username"/></h4>

</body>
</html>
```

# 5. struts.xml

所有类注解无需创建 struts.xml 文件。

# 6. web.xml

只要创建一个典型的web.xml文件，并声明FilterDispatcher过滤器标准。

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Struts 2 Web Application</display-name>

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</f
  </filter>

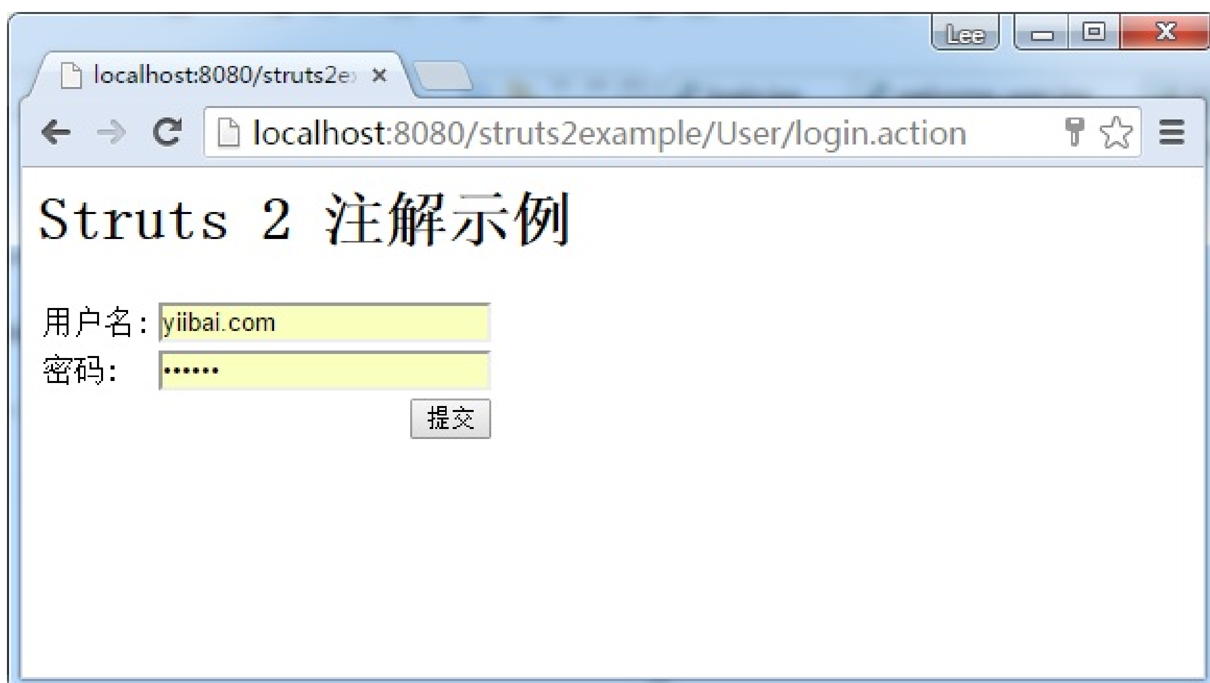
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

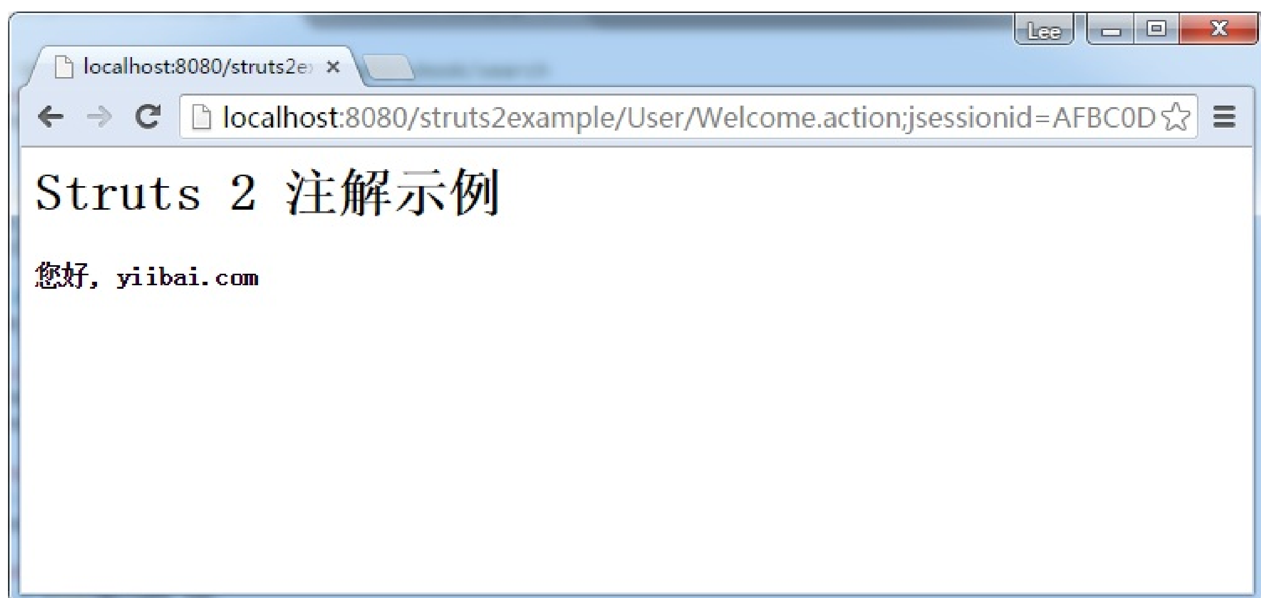
## 7. 运行测试

LoginAction.action 改为 login.action，请参阅上面的“命名转换器”。

http://localhost:8080/struts2example/User/login.action




提交到 http://localhost:8080/Struts2Example/User/Welcome.action 后显示：




## 参考

1. Struts 2 约定插件文档
2. Strust 2 Hello World (XML 版本)

# Struts2 @ResultPath注释示例 - Struts2教程

在Struts 2中, @ResultPath 注解用于控制Struts2找到存储的结果或JSP页面。默认情况下，它会找到结果页在 “WEB-INF/content/” 文件夹。不知道为什么在Struts2注解设置 “WEB-INF/content/” 作为默认目录, 但是大部分的应用并不将结果页放入到 “WEB-INF/content/” 目录. 可能Struts2惯例也并不是一个标准的文件夹结构。 我一般是在 Struts 2 根路径作为默认的文件夹。

## @ResultPath 示例

## 1. 默认结果路径

在登录动作类,设置 “/User” 作为命名空间, 并重定向到 “pages/login.jsp” 页面。

P.S 假设struts2example是上下文servlet名称

```
@Namespace("/User")
@Result(name="success",location="pages/login.jsp")
public class LoginAction extends ActionSupport{
}
```

访问它，如下：

```
http://localhost:8080/struts2example/User/login.action
```

Struts 2将从默认位置找到“login.jsp”结果页面：

```
/struts2example/WEB-INF/content/User/pages/login.jsp
```

## 2. 定制结果路径

如果JSP结果页面存储在其他位置，那么可以使用 @ResultPath注释设置改变它。

```
@Namespace("/User")
@ResultPath(value="/")
@Result(name="success",location="pages/login.jsp")
public class LoginAction extends ActionSupport{
}
```

再一次访问：

```
http://localhost:8080/struts2example/User/login.action
```

现在Struts2将从不同的位置找到“login.jsp”结果页面：

```
/Struts2Example/User/pages/login.jsp
```

## 全局@ResultPath

@ResultPath只适用于类级别。在全局范围内应用它，可以在 struts.xml 文件中进行配置。

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.convention.result.path" value="/"/>
</struts>
```

## 参考

1. Struts 2 @ResultPath 注解文档

# Struts2 include(包含)多个配置文件 - Struts2教程

Struts 2自带有“包含文件”功能，包含多个Struts配置文件合并为一个单元。

## 单个Struts配置文件

让我们来看看一个糟糕的 Struts 2 配置示例。

**struts.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="default" namespace="/" extends="struts-default">
</package>

<package name="audit" namespace="/audit" extends="struts-default">
    <action name="WelcomeAudit">
        <result>pages/welcome_audit.jsp</result>
    </action>
</package>

<package name="user" namespace="/user" extends="struts-default">
    <action name="WelcomeUser">
        <result>pages/welcome_user.jsp</result>
    </action>
</package>

</struts>
```

在上面的Struts配置文件中，组织所有“用户”和“审核”配置设置在一个文件中，这不是建议的，必须回避。应该打破这种形式，而将struts.xml文件分成更小的模块相关的部分。

## 多个Struts配置文件

在Struts2，应该给每个模块一个Struts配置文件。在这种情况下，可以创建三个文件：

1. audit-struts.xml – 将所有审计模块设置在这里。

2. user-struts.xml – 将所有用户模块设置在这里。
3. struts.xml – 默认设置，包含 struts-audit.xml 和 Struts-user.xml 两个文件。

**struts-audit.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="audit" namespace="/audit" extends="struts-default">
    <action name="WelcomeAudit">
        <result>pages/welcome_audit.jsp</result>
    </action>
</package>

</struts>
```

**struts-user.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="user" namespace="/user" extends="struts-default">
    <action name="WelcomeUser">
        <result>pages/welcome_user.jsp</result>
    </action>
</package>

</struts>
```

**struts.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

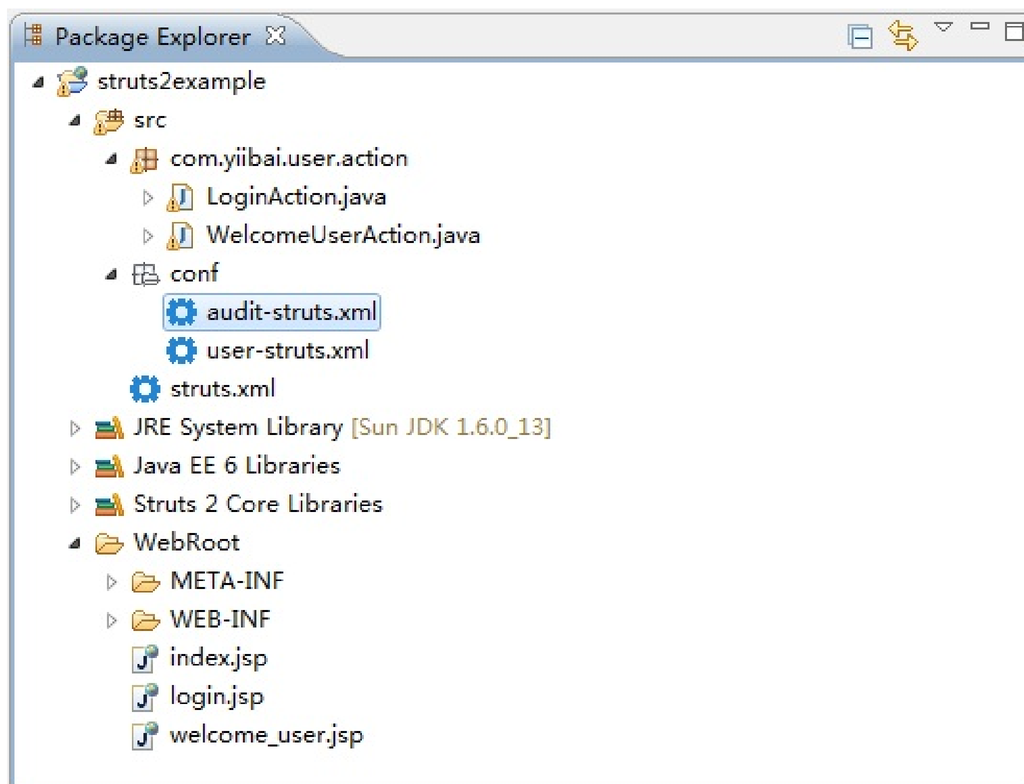
<struts>

<package name="default" namespace="/" extends="struts-default">
</package>

<include file="conf/user-struts.xml"></include>
<include file="conf/audit-struts.xml"></include>

</struts>
```

现在文件夹结构看起来如下：



# Struts2命名空间配置和解释 - Struts2教程

Struts 2的命名空间是一个新的概念，用来处理多个模块。由下式给出一个命名空间的每个模块。此外，它还可以用来避免位于不同的模块相同的操作名称之间的冲突。

看下面的一张图来了解一个URL匹配Struts 2的动作命名空间。




## 1. 命名空间配置

让我们通过一个Struts2的命名空间配置的例子来了解它是如何与URL和文件夹相匹配。

P.S 包中的“name”不会影响结果，只是给一个有意义的名字。

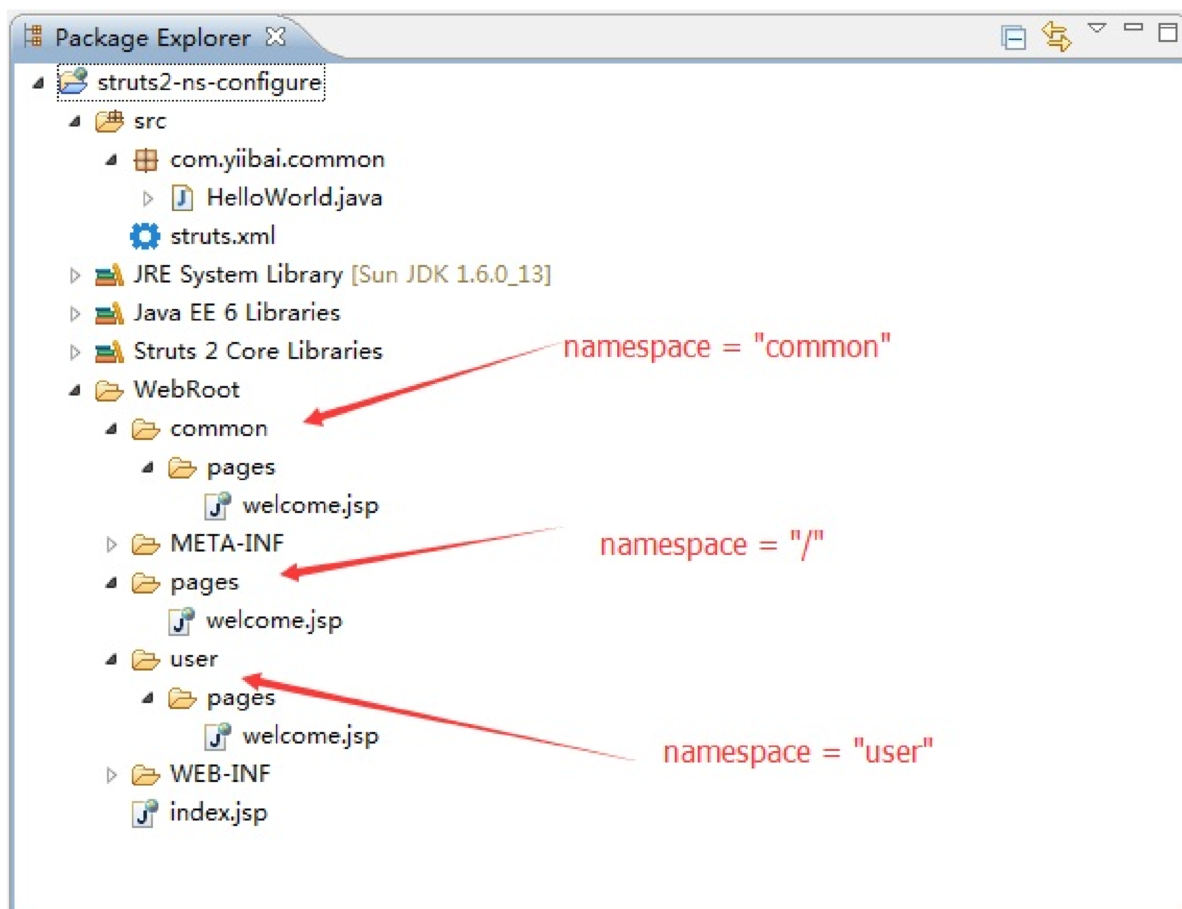
**struts.xml**

```
**<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
<struts>
    <package name="default" namespace="/" extends="struts-default">
        <action name="SayWelcome">
            <result>/pages/welcome.jsp</result>
        </action>
    </package>

    <package name="common" namespace="/common" extends="struts-defa
        <action name="SayWelcome">
            <result>/common/pages/welcome.jsp</result>
        </action>
    </package>

    <package name="user" namespace="/user" extends="struts-default'
        <action name="SayWelcome">
            <result>/common/user/welcome.jsp</result>
        </action>
    </package>
</struts>**
```

Struts 2的动作命名空间映射到文件夹结构。



## 2. JSP视图页面

3 JSP页面视图具有相同的文件名，但是在不同的模块位置。

根 – pages/welcome.jsp

```
<html>
<head>
<title>Struts2命名空间示例 - yiibai.com</title>
</head>
<body>
    <h1>Struts2命名空间示例</h1>
    <h4>Welcome - namespace = "root"</h4>
</body>
</html>
```

Common 模块 – common/pages/welcome.jsp

```
<html>
<head>
<title>Struts2命名空间示例 - yiibai.com</title>
</head>
<body>
    <h1>Struts2命名空间示例</h1>
    <h4>Welcome - namespace = "common"</h4>
</body>
</html>
```

User 模块 – user/pages/welcome.jsp

```
<html>
<head>
<title>Struts2命名空间示例 - yiibai.com</title>
</head>
<body>
    <h1>Struts2命名空间示例</h1>

    <h4>Welcome - namespace = "user"</h4>
</body>
</html>
```

## 3. 映射 – 如何工作?

示例 **1** URL : http://localhost:8080/Struts2Example/SayWelcome.action

将匹配根命名空间。

```
    <package name="default" namespace="/" extends="struts-default">
        <action name="SayWelcome">
            <result>/pages/welcome.jsp</result>
        </action>
    </package>
```

这会显示 pages/welcome.jsp 页面的内容

示例 **2** URL : http://localhost:8080/Struts2Example/common/SayWelcome.action
这会匹配 common 命名空间的内容：

```
    <package name="common" namespace="/common" extends="struts-defa
        <action name="SayWelcome">
            <result>/common/pages/welcome.jsp</result>
        </action>
    </package>
```

这会显示 ****common/pages/welcome.jsp** **页面的内容

示例 **3** URL : http://localhost:8080/Struts2Example/user/SayWelcome.action 这会匹配 common 命名空间的内容：

```
<package name="user" namespace="/user" extends="struts-default">
    <action name="SayWelcome">
        <result>/common/user/welcome.jsp</result>
    </action>
</package>
```

这会显示 **user/pages/welcome.jsp** 页面的内容.

# 参考

1. Struts2命名空间配置参考

代码下载：http://pan.baidu.com/s/1hqe1nZe

# Struts2开发者模式 - Struts2教程

在Struts2开发中，这应该是第一个学习配置的值。为了启用 Struts 2 的开发模式，可以通过自动配置显著增加Struts2的开发速度和属性文件加载，以及额外的日志和调试功能。

注：自动重新加载功能真的是一个方便的功能。每次修改属性或***XML***配置文件更改，应用程序不再需要重启才能生效。默认情况下，*Struts 2*的开发模式是禁用的。

## 启用Struts2开发模式

将“struts.devMode”的值设置为true，可以在Struts的属性文件或XML配置文件。

**struts.properties**

```
struts.devMode = true
```

**struts.xml**

```
<struts>
     <constant name="struts.devMode" value="true" />
</struts>
```

## 禁用Struts 2的开发模式

设置“struts.devMode”为false，无论是在Struts属性文件或XML配置文件。

**struts.properties**

```
struts.devMode = false
```

**struts.xml**

```
<struts>
     <constant name="struts.devMode" value="false" />
</struts>
```

开发模式只适合于开发和调试环境。在生产环境中，你必须禁用它。因为整个应用程序的配置它会引起对性能显著影响，属性文件将在每次请求重新加载，许多额外的日志和调试信息也将提供。

## 参考

1. Struts 2 开发模式文档

# 如何删除Struts2动作的后缀扩展名 - Struts2教程

在Struts2中，所有动作类有一个默认的后缀 .action 扩展。 例如，

```
<struts>
  <package name="default" namespace="/" extends="struts-default">
    <action name="SayStruts2">
        <result>pages/printStruts2.jsp</result>
    </action>
  </package>
</struts>
```

如要访问“SayStruts2”动作类，需要使用以下网址：

```
Action URL : http://localhost:8080/Struts2Example/SayStruts2.action
```

## 配置动作扩展

Struts 2是允许配置扩展名的，要对其进行更改，只需要声明一个常数“struts.action.extension”值：

## 1. html 扩展

更改动作类为 .html 的扩展名。

```
<struts>

  <constant name="struts.action.extension" value="html"/>

  <package name="default" namespace="/" extends="struts-default">
    <action name="SayStruts2">
        <result>pages/printStruts2.jsp</result>
    </action>
  </package>

</struts>
```

现在，可以通过访问“SayStruts2”动作类，使用如下URL：

```
Action URL : http://localhost:8080/Struts2Example/SayStruts2.html
```

## 2. 不使用扩展

动作类更改为空的扩展。

```
<struts>

  <constant name="struts.action.extension" value=""/>

  <package name="default" namespace="/" extends="struts-default">
    <action name="SayStruts2">
        <result>pages/printStruts2.jsp</result>
    </action>
  </package>

</struts>
```

现在，可以通过如下的URL来访问“SayStruts2' 动作类 :

```
Action URL : http://localhost:8080/Struts2Example/SayStruts2
```

# 使用Struts2动作 - Struts2教程

在Struts2中，会花大部分的时间用在用动作来处理工作。动作类包含业务逻辑，获取资源包，保存数据，验证，并选择应发回给用户的视图的结果页面。这是Struts2的核心，所以必须要了解动作的基本概念。

## 1. 动作 - Action

Struts 2动作不强迫你实现任何接口或扩展类，它只是需要你实现一个 execute()方法返回一个字符串来表示其应该返回的结果页面。

```
package com.yiibai.user.action;
public class LoginAction{
    //business logic
    public String execute() {
        return "success";
    }
}
```

在struts.xml中，配置使用Action类动作(action)标记和类属性。定义结果页面用结果的标签和动作名称返回给用户，可以用它来访问动作类的名称属性。

```
<package name="user" namespace="/User" extends="struts-default">
  <action name="validateUser" class="com.yiibai.user.action.LoginAc
    <result name="success">pages/welcome.jsp</result>
  </action>
<package>
```

现在，可以通过 .action 扩展名后缀访问动作。

```
http://localhost:8080/Struts2Example/User/validateUser.action
```

默认.action是可配置的，只需要设置“struts.action.extension”的值，以满足您的需要。

## 2. 可选动作接口

Struts 2带有一个可选的动作接口(com.opensymphony.xwork2.Action)。通过实现这个接口，它带来了一些方便和好处，看下面的源代码：

```
package com.opensymphony.xwork2;
public interface Action {
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";
    public String execute() throws Exception;
}
```

这个接口是非常简单的，配有5常用常数值：**success, error, none, input and logic**。现在的动作类可以直接使用常量。

```
package com.yiibai.user.action;
import com.opensymphony.xwork2.Action;
public class LoginAction{
    //business logic
    public String execute() {
        return SUCCESS;
    }
}
```

不明白为什么很多Struts开发人员喜欢实现此动作接口，它更好地扩展了ActionSupport。

# 3. ActionSupport

Support 类，通常的做法是提供接口的默认实现。

ActionSupport (**com.opensymphony.xwork2.ActionSupport**), 一个非常强大和方便的类，它提供了几个重要接口的缺省实现：

```
public class ActionSupport implements Action, Validateable,
   ValidationAware, TextProvider, LocaleProvider, Serializable {
 ...
}
```

ActionSupport 类提供一些功能：

1. 验证 – 声明一个validate()方法，并在里面实现验证代码。

  1. 文字本地化 – 使用gettext()方法来获得资源包的消息。

```
package com.yiibai.user.action;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport{
    private String username;
    private String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    //business logic
    public String execute() {
        return "SUCCESS";
    }
        //simple validation
    public void validate(){
        if("".equals(getUsername())){
            addFieldError("username", getText("username.required")
        }
        if("".equals(getPassword())){
            addFieldError("password", getText("password.required")
        }
    }
}
```

在大多数情况下，应该扩展此类妥当，方便提供功能，除非你有理由不这样做。这也是一个很不错的学习类，以了解如何做一些重要的Struts2接口的实现。

## 4. 动作注释

Struts 2对注解有很好的支持，你可以摆脱XML文件，并使用@action在动作类上替换。

```
package com.yiibai.user.action;
import org.apache.struts2.convention.annotation.Action;
import org.apache.struts2.convention.annotation.Namespace;
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.ResultPath;
import com.opensymphony.xwork2.ActionSupport;
@Namespace("/User")
@ResultPath(value="/")
public class ValidateUserAction extends ActionSupport{
    @Action(value="Welcome", results={
        @Result(name="success",location="pages/welcome_user.jsp")
    })
    public String execute() {
        return SUCCESS;
    }
}
```

## 总结

扩展ActionSupport类，它适合在大多数情况下。

# Struts2的ActionError＆ActionMessage示例 - Struts2教程

本教程显示使用Struts2的 ActionError 和 ActionMessage 类。

1. ActionError – 是用来发送错误信息反馈给用户 - 通过 <s:actionerror/> 来显示。

```
<s:if test="hasActionErrors()">
   <div class="errors">
      <s:actionerror/>
   </div>
</s:if>
```

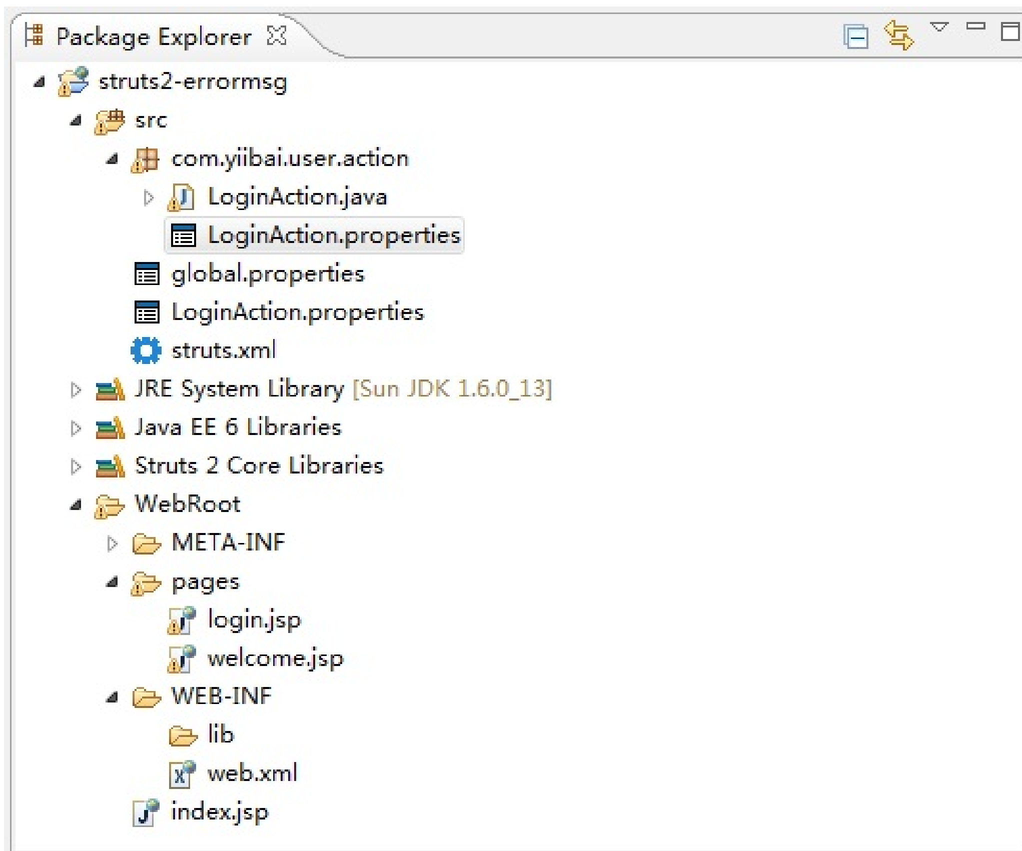
1. **ActionMessage** – 用于发送信息的反馈消息给用户，通过 **<s:actionmessage/>** 来显示。

```
<s:if test="hasActionMessages()">
   <div class="welcome">
      <s:actionmessage/>
   </div>
</s:if>
```

这里有一个简单的登录表单，如果用户名不等于“yiibai.com”将显示错误消息(actionerror)，否则重定向到另一个页面，显示欢迎信息(ActionMessage)。此外，所有的标签和错误消息检索来自资源包(属性文件)。

## 1. 文件夹结构

在MyEclipse中创建一个web工程，名称为：struts2-errormsg，看这个项目结构，如下图：




## 2. 属性文件

一共有两个属性文件用来存储信息，其中 LoginAction.properties 文件放在 com.yiibai.user.action 包下。

LoginAction.properties

```
#Welcome messages
welcome.hello = 你好
#error message
username.required = 用户名不可以为空
password.required = 密码不可以为空
```

global.properties

```
#Global messages
global.username = 用户名
global.password = 密码
global.submit = 提交
global.reset = 重置
```

## 3. 动作-Action

一个经典的动作类，做一个简单的检查，以确认用户名是否等于“yiibai.com"，并使用 addActionError()设置错误信息或addActionMessage()设置成功的消息。

```
package com.yiibai.user.action;

import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport{

    private String username;
    private String password;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    //business logic
    public String execute() {

        return "SUCCESS";

    }

    //simple validation
    public void validate(){
        if("yiibai.com".equals(getUsername())){
            addActionMessage("You are valid user!");
        }else{
            addActionError("I don't know you, dont try to hack me!'
        }
    }
}
```

## 4. JSP页面视图

两个简单的JSP页面以及CSS样式自定义错误消息。

login.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts2 ActionError & ActionMessage 示例</title>
<style type="text/css">
.errors {
    background-color:#FFCCCC;
    border:1px solid #CC0000;
    width:400px;
    margin-bottom:8px;
}
.errors li{
    list-style: none;
}
</style>
</head>
<body>
<h1>Struts2 ActionError & ActionMessage 示例</h1>
<s:if test="hasActionErrors()">
   <div class="errors">
      <s:actionerror/>
   </div>
</s:if>
<s:form action="validateUser">
    <s:textfield key="global.username" name="username"/>
    <s:password key="global.password" name="password"/>
    <s:submit key="global.submit" name="submit"/>
</s:form>
</body>
</html>
```

**welcome.jsp**

```
**<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>Struts2 ActionError & ActionMessage 示例</title>
<style type="text/css">
.welcome {
    background-color: #DDFFDD;
    border: 1px solid #009900;
    width: 200px;
}
.welcome li {
    list-style: none;
}
</style>
</head>
<body>
    <h1>Struts 2 ActionError & ActionMessage示例</h1>
    <s:if test="hasActionMessages()">
        <div class="welcome">
            <s:actionmessage />
        </div>
    </s:if>
    <h4>
        <s:property value="getText('welcome.hello')" />
        <s:property value="username" />
    </h4>
</body>
</html>**
```

# 5. struts.xml

链接所有的在一起

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
  <constant name="struts.custom.i18n.resources" value="global" />

  <package name="user" namespace="/user" extends="struts-default">
     <action name="login">
     <result>/pages/login.jsp</result>
     </action>
     <action name="validateUser" class="com.yiibai.user.action.Log:
     <result name="SUCCESS">/pages/welcome.jsp</result>
     <result name="input">/pages/login.jsp</result>
     </action>
   </package>

</struts>
```

在Struts2，ActionError和ActionMessage功能和用法与Struts1非常相似。

## 6. 运行并测试

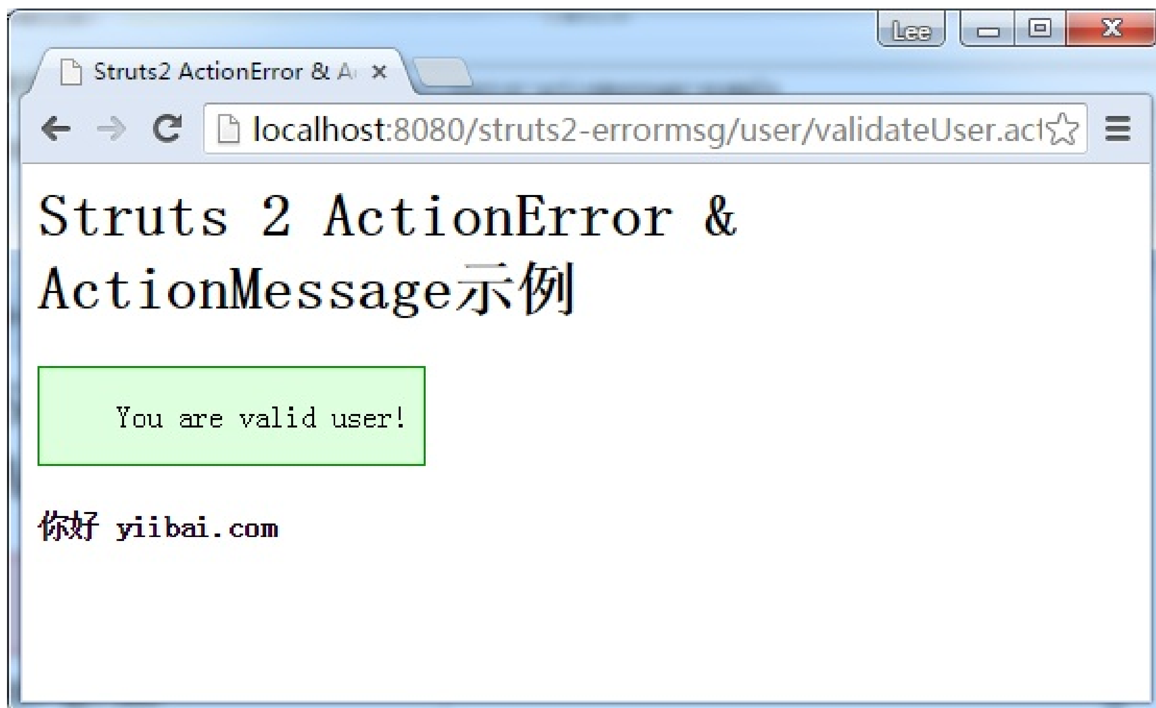
http://localhost:8080/struts2-errormsg/user/login.action




用户名是无效的，显示错误信息：<s:actionerror/>




用户名是有效的，显示欢迎信息：<s:actionmessage/>




源代码下载 – Struts2-ActionError-ActionMessage.zip

# Struts2模型驱动实例 - Struts2教程

这里我们创建一个web工程为：struts2-modeldrive， 用于讲解演示Struts2模型驱动这一章内容的学习。

如果一个动作实现了“模型驱动”- ModelDriven 接口，它就获得了表单数据自动传输到对象的额外能力。请参见下面的完整的例子：

## 1. 域对象

一个顾客(customer)对象，有 setter 和 getter 方法。

**Customer.java**

```
package com.yiibai.common;

public class Customer{

    String name;
    int age;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

## 2. 动作 - Action

Action类，实现了模型驱动ModelDriven 接口，声明getModel()方法返回客户的对象。当表单数据提交到这个动作，它会自动将表单数据传输到客户的属性。

客户对象必须手动初始化。

**CustomerAction.java**

```
package com.yiibai.common.action;

import com.yiibai.common.Customer;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

public class CustomerAction extends ActionSupport
    implements ModelDriven{

    //have to initialize it
    Customer customer = new Customer();

    public String execute() throws Exception {

        return SUCCESS;

    }

    public Object getModel() {

        return customer;

    }
}
```

## 3. JSP页面

JSP页面的模型驱动(ModelDriven)的示范。

**addCustomer.jsp**

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>

<body>
<h1>Struts 2 ModelDriven example</h1>

<h2>Add Customer</h2>
<s:form  action="customerAction" >
  <s:textfield name="name" label="Name" />
  <s:textfield name="age" label="Age" value=""/>
  <s:submit />
</s:form>

</body>
</html>
```

### success.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>

<body>
<h1>Struts 2 ModelDriven example</h1>

<h2>Customer Details</h2>
Name : <s:property value="name" /><br>
Age : <s:property value="age" /><br>

</body>
</html>
```

# 4. struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
     <constant name="struts.devMode" value="true" />

    <package name="default" namespace="/" extends="struts-default">

        <action name="addCustomerAction"
            class="com.yiibai.common.action.CustomerAction" >
            <result name="success">pages/addCustomer.jsp</result>
        </action>

        <action name="customerAction"
            class="com.yiibai.common.action.CustomerAction" >
            <result name="success">pages/success.jsp</result>
        </action>

    </package>

</struts>
```

# 5. 示例

访问客户表，填写表格 (name : “yiibai.com”, age ” “26”) 并点击提交按钮，表单数据(name & age) 将自动转移到客户的属性(name & age) (按属性名称匹配)。

http://localhost:8080/struts2-modeldrive/addCustomerAction.action

http://localhost:8080/struts2-modeldrive/customerAction.action

工程源代码下载 - http://pan.baidu.com/s/1hqxyjf2

# Struts2映射拦截动作 - Struts2教程

Struts 2的开发者用来声明行为属于一个包，扩展 “struts-default“, 其中包含默认设置的拦截。

```
<package name="default" namespace="/" extends="struts-default">
    <action name="testingAction"
        class="com.yiibai.common.action.TestingAction" >
        <result name="success">pages/result.jsp</result>
    </action>
</package>
```

拦截器的默认设置进行分组为“defaultStack”在struts-default.xml文件中，它位于struts2-core.jar 文件，“defaultStack”提供所有的核心Struts2功能，这是最适合应用的需要。

试着学习struts-default.xml文件，它总是最好的拦截器的参考。

## 映射拦截动作

为了其它的拦截器映射到动作，请使用“interceptor-ref”元素。

```
<package name="default" namespace="/" extends="struts-default">
    <action name="testingAction"
        class="com.yiibai.common.action.TestingAction" >
        <interceptor-ref name="timer"/>
        <interceptor-ref name="logger"/>
        <result name="success">pages/result.jsp</result>
    </action>
</package>
```

在上面的代码片段，将其映射“timer” 和 “logger”通过“interceptor-ref”元素拦截到“TestingAction”动作类。拦截器会按它们声明的顺序触发。

由于“TestingAction”它声明自己的拦截器，它的直接失去拦截器的所有继承默认设置，你必须明确才能使用它，见下面声明“defaultStack”的例子。

```
<package name="default" namespace="/" extends="struts-default">
    <action name="testingAction"
        class="com.yiibai.common.action.TestingAction" >
        <interceptor-ref name="timer"/>
        <interceptor-ref name="logger"/>
        <interceptor-ref name="defaultStack"/>
        <result name="success">pages/result.jsp</result>
    </action>
</package>
```

## 参考

1. Struts 2 拦截器文档

# Struts2重写拦截器参数 - Struts2教程

在Struts2中，可以设置或通过普通的标签重写拦截器的参数。见下面的例子：

```
<package name="default" namespace="/" extends="struts-default">
   <action name="whateverAction"
    class="com.yiibai.common.action.WhateverAction" >
    <interceptor-ref name="workflow">
        <param name="excludeMethods">whateverMethod</param>
    </interceptor-ref>
    <result name="success">pages/whatever.jsp</result>
   </action>
</package>
```

然而，在上面的代码片段，动作类被声明为自己的拦截器， 它会导致继承“defaultStack”拦截器的直接丢失。

如果你想保持“defaultStack”拦截器，并覆盖工作流的excludeMethods参数呢？没问题，试试这个：

```
<package name="default" namespace="/" extends="struts-default">
   <action name="whateverAction"
    class="com.yiibai.common.action.WhateverAction" >
    <interceptor-ref name="defaultStack">
        <param name="workflow.excludeMethods">whateverMethod</param
    </interceptor-ref>
    <result name="success">pages/whatever.jsp</result>
   </action>
</package>
```

上面的代码片段将保持“defaultStack”拦截并覆盖“workflow”参数。

## 参考

1. Struts2拦截器文档
2. Struts2流程拦截器文档

# Struts2拦截器栈的例子 - Struts2教程

很多时候，相同的一组拦截器可以适用于不同的动作类，例如，

```
<package name="default" namespace="/" extends="struts-default">

    <action name="checkInAction"
    class="com.yiibai.common.action.CheckInAction" >
    <interceptor-ref name="timer"/>
        <interceptor-ref name="logger"/>
    <interceptor-ref name="defaultStack" />
    <result name="success">/pages/checkIn.jsp</result>
    </action>

    <action name="checkOutAction"
    class="com.yiibai.common.action.CheckOutAction" >
    <interceptor-ref name="timer"/>
        <interceptor-ref name="logger"/>
    <interceptor-ref name="defaultStack" />
    <result name="success">/pages/checkOut.jsp</result>
    </action>

</package>
```

在上述情况下，它有许多重复工作以及不能重复使用。

幸运的是，在Struts 2自带的拦截器栈，使开发人员建立一组拦截到一个单元名为“栈名字”， 和可以通过“栈名字”引用操作它。

最佳做法建议组合相同的一组拦截器到一个拦截器栈摆脱重复的工作，并增加了项目的可重用性。

```
<package name="default" namespace="/" extends="struts-default">

     <interceptors>
           <interceptor-stack name="defaultStackWithLog">
             <interceptor-ref name="timer"/>
             <interceptor-ref name="logger"/>
         <interceptor-ref name="defaultStack" />
        </interceptor-stack>
    </interceptors>

    <action name="checkInAction"
    class="com.yiibai.common.action.CheckInAction" >
    <interceptor-ref name="defaultStackWithLog"/>
    <result name="success">/pages/checkIn.jsp</result>
    </action>

    <action name="checkOutAction"
    class="com.yiibai.common.action.CheckOutAction" >
    <interceptor-ref name="defaultStackWithLog"/>
    <result name="success">/pages/checkOut.jsp</result>
    </action>

</package>
```

在上面的例子更新，声明一个拦截器栈，命名为“defaultStackWithLog”其中包括“timer“, “logger” 和 “defaultStack” 拦截器，并且它通过“interceptor-ref”元素引用一个正常的拦截器。

## 参考

1. Struts2拦截器文档

# Struts2 execAndWait拦截器例子 - Struts2教程

在Struts2中附带一个名为"execAndWait"一个非常有趣的"执行和等待"拦截器，这是一个非常方便的拦截器长时间运行操作在后台，显示用户的自定义的等待页面。在本教程中，它显示了一个完整的使用 Struts2 execAndWait 拦截器的例子。

## 1. 动作

一个普通的动作类，有一个长时间运行进程，证明了execAndWait效果。

**LongProcessAction.java**

```
package com.yiibai.common.action;

import com.opensymphony.xwork2.ActionSupport;

public class LongProcessAction extends ActionSupport{

    public String execute() throws Exception {

        //it should be delay few seconds,
        //unless you have a super powerful computer.
        for(int i =0; i<1000000; i++){
            System.out.println(i);
        }
        return SUCCESS;

    }
}
```

## 2. JSP页面

创建两个页面：

1. wait.jsp - 显示给用户，长时间运行的进程。
2. success.jsp - 显示给用户的过程完成之后。

HTML meta refresh 记得把元刷新的等待页面顶部; 否则，该网页将不重定向到成功页面，即使该过程完成。

在这个wait.jsp，元刷新设置在每5秒网页重新加载，如果该过程完成后，将重定向到 success.jsp, 否则留在同一个页面。

**wait.jsp**

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 execAndWait 示例</title>
<meta http-equiv="refresh" content="5;url=<s:url includeParams="al
</head>

<body>
<h1>Struts 2 execAndWait 示例</h1>

<h3>Please wait while we process your request...</h3>

</body>
</html>
```

**success.jsp**

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 execAndWait 示例</title>
</head>

<body>
<h1>Struts 2 execAndWait 示例</h1>

<h3>Done</h3>

</body>
</html>
```

# 3. 执行和等待拦截器

链接动作类并声明"execAndWait"拦截器。execAndWait 参数

1. delay (optional) : 以毫秒为单位初始延迟显示在wait.jsp。默认是没有延迟的。
2. delaySleepInterval (optional) : 时间间隔是以毫秒为单位来检查后台进程是否已经完成，默认值是100毫秒。

**struts.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
     <constant name="struts.devMode" value="true" />
    <package name="default" namespace="/" extends="struts-default">
        <action name="longProcessAction"
            class="com.yiibai.common.action.LongProcessAction" >

            <interceptor-ref name="execAndWait">
                <param name="delay">1000</param>
                <param name="delaySleepInterval">500</param>
            </interceptor-ref>

            <result name="wait">/pages/wait.jsp</result>
            <result name="success">/pages/success.jsp</result>
        </action>

    </package>
</struts>
```

在这种情况下，将延迟1秒显示至wait.jsp，并检查后台进程是否在每500毫秒完成。即使这个过程完成后，它仍然需要等待 wait.jsp 元刷新来触发页面重载。

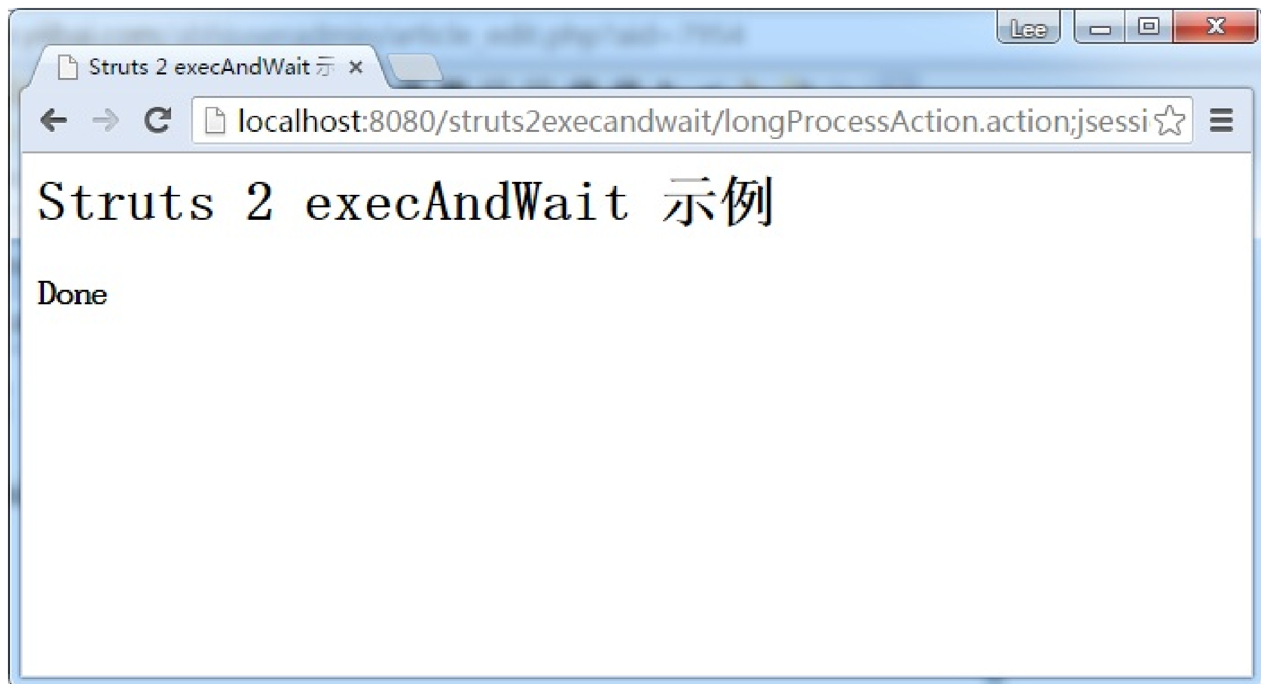
## 4. 示例

访问网址： http://localhost:8080/struts2execandwait/longProcessAction.action



延时1秒，显示在 wait.jsp 。

当该过程完成时，自动显示在 success.jsp。




代码下载：http://pan.baidu.com/s/1o62BHGY

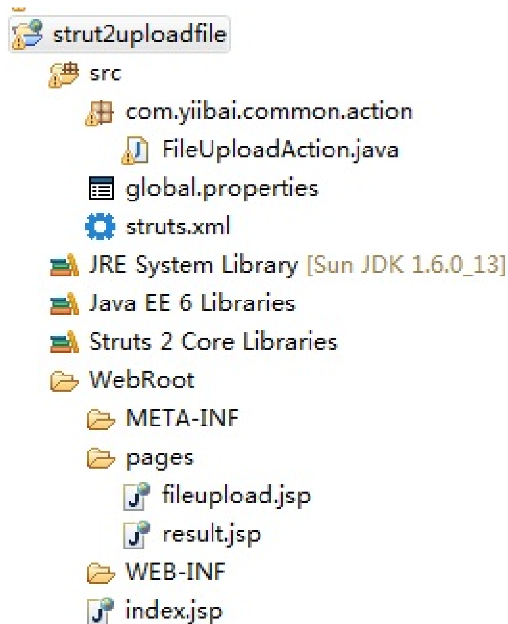
## 参考

1. Struts2 execAndWait拦截器文档
2. HTML Meta Refresh

# Struts2文件上传例子 - Struts2教程

在Struts2， <s:file> 标签用于创建一个HTML文件上传组件，允许用户从本地磁盘选择文件，并将其上传到服务器。在本教程中，您将创建与文件上传组件JSP页面，设置最大大小和允许上传文件的内容类型，并显示上传文件的详细信息。

这里创建一个Web工程：strut2uploadfile，来演示在多个复选框如何设置的默认值，整个项目的结构如下图所示：




## 1. 动作类

Action类的文件上传，声明“File”变量来存储用户上传的文件，两个字符串变量以存储文件名和内容类型。“文件上传拦截器”通过设置“X”的ContentType()，并设置“X”FileName()会自动注入上传的文件细节，确保方法名拼写正确。

P.S X是以存储上传的文件中的变量。

文件上传功能是依赖于“文件上传拦截器”，确保将其纳入行动的堆栈。幸运的是，默认的堆栈已经包含了“文件上传拦截器”。

**FileUploadAction.java**

```
package com.yiibai.common.action;

import java.io.File;

import com.opensymphony.xwork2.ActionSupport;

public class FileUploadAction extends ActionSupport{

    private File fileUpload;
    private String fileUploadContentType;
    private String fileUploadFileName;

    public String getFileUploadContentType() {
        return fileUploadContentType;
    }

    public void setFileUploadContentType(String fileUploadContentTy
        this.fileUploadContentType = fileUploadContentType;
    }

    public String getFileUploadFileName() {
        return fileUploadFileName;
    }

    public void setFileUploadFileName(String fileUploadFileName) {
        this.fileUploadFileName = fileUploadFileName;
    }

    public File getFileUpload() {
        return fileUpload;
    }

    public void setFileUpload(File fileUpload) {
        this.fileUpload = fileUpload;
    }

    public String execute() throws Exception{

        return SUCCESS;

    }

    public String display() {
        return NONE;
    }

}
```

## 2. 结果页面

使用<s:file>标签来渲染一个文件上传组件，并设置表单的enctype类型为："multipart/form-data"。

**fileupload.jsp**

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<s:head />
</head>

<body>
<h1>Struts 2 <s:file> file upload example</h1>

<s:form action="resultAction" namespace="/"
method="POST" enctype="multipart/form-data">

<s:file name="fileUpload" label="Select a File to upload" size="40'

<s:submit value="submit" name="submit" />

</s:form>

</body>
</html>
```

**result.jsp**

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>

<body>
<h1>Struts 2 <s:file> file upload example</h1>

<div><div class="ads-in-post hide_if_width_less_800">
<script async src="//pagead2.googlesyndication.com/pagead/js/adsbyg
<!-- 728x90 - After2ndH4 -->
<ins class="adsbygoogle hide_if_width_less_800"
     style="display:inline-block;width:728px;height:90px"
     data-ad-client="ca-pub-2836379775501347"
     data-ad-slot="3642936086"
     data-ad-region="yiibairegion"></ins>
<script>
(adsbygoogle = window.adsbygoogle || []).push({});
</script>
</div></div><h2>
   File Name : <s:property value="fileUploadFileName"/>
</h2>

<h2>
   Content Type : <s:property value="fileUploadContentType"/>
</h2>

<h2>
   File : <s:property value="fileUpload"/>
</h2>

</body>
</html>
```

# 3. struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

     <constant name="struts.devMode" value="true" />
     <constant name="struts.custom.i18n.resources" value="global" /

    <package name="default" namespace="/" extends="struts-default">

    <action name="fileUploadAction"
        class="com.yiibai.common.action.FileUploadAction" method="d
        <result name="none">/pages/fileupload.jsp</result>
    </action>

    <action name="resultAction" class="com.yiibai.common.action.Fil
        <interceptor-ref name="exception"/>
            <interceptor-ref name="i18n"/>
            <interceptor-ref name="fileUpload">
               <param name="allowedTypes">text/plain</param>
               <param name="maximumSize">10240</param>
          </interceptor-ref>
            <interceptor-ref name="params">
                <param name="excludeParams">dojo\..*,^struts\..*</p
            </interceptor-ref>
            <interceptor-ref name="validation">
                <param name="excludeMethods">input,back,cancel,brow
            </interceptor-ref>
            <interceptor-ref name="workflow">
                <param name="excludeMethods">input,back,cancel,brow
            </interceptor-ref>

        <result name="success">/pages/result.jsp</result>
        <result name="input">/pages/fileupload.jsp</result>

    </action>
   </package>
</struts>
```

文件大小限制 在这个例子中，您将通过“文件上传拦截”上传文件大小的限制，该值以字节为单位计数。在本实例中，上载文件的最大尺寸是10KB。

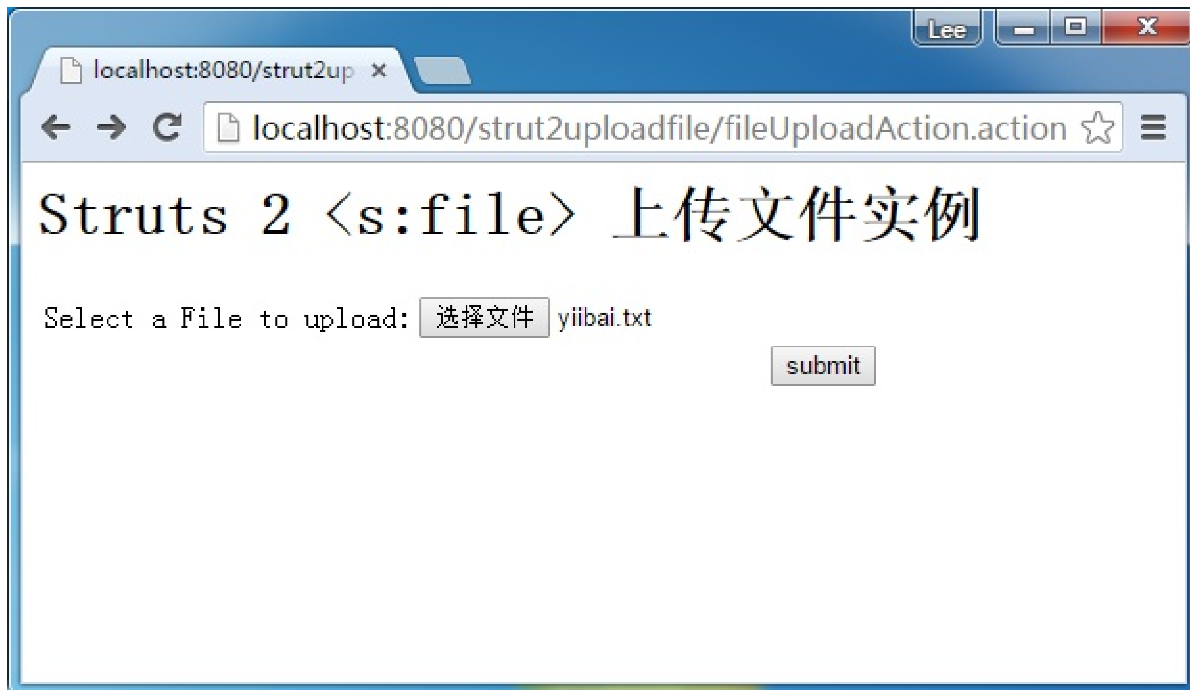
注：上传文件的默认最大文件大小为2MB

文件类型 可以通过设置“文件上传拦截器”允许的文件类型。在这种情况下，上传文件只接受“text/plain”的类型。

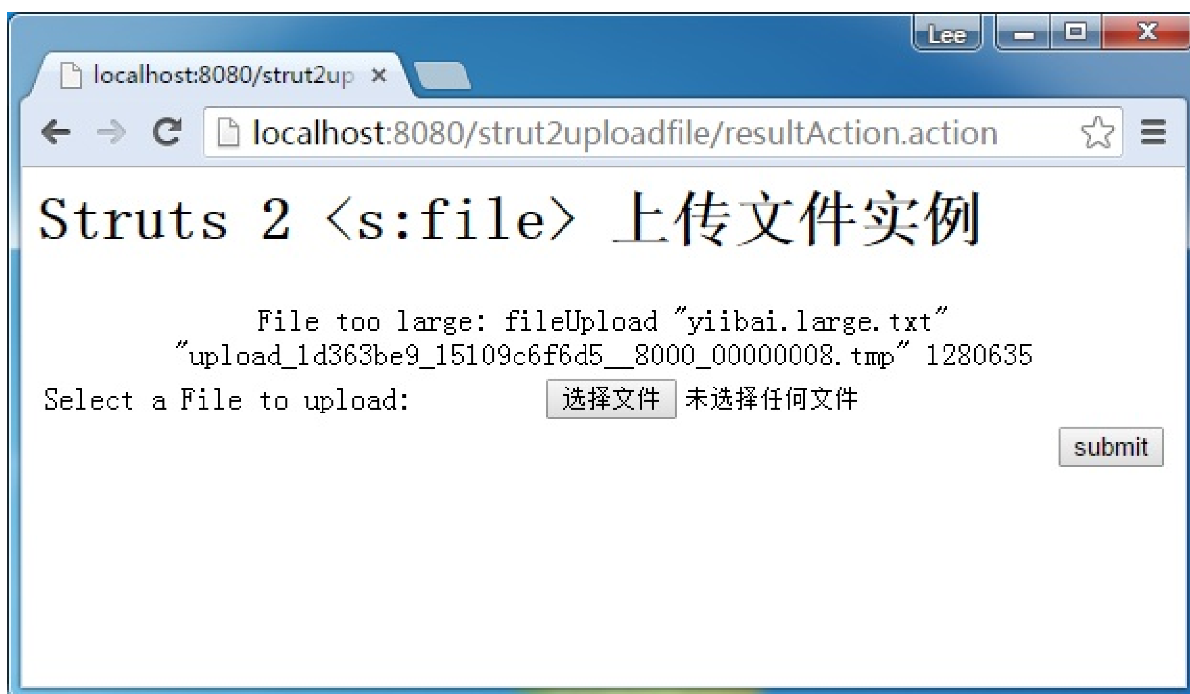
在Struts2中，有好几种方面做到这一点，查看Struts2的文件上传文档。

# 4. 示例

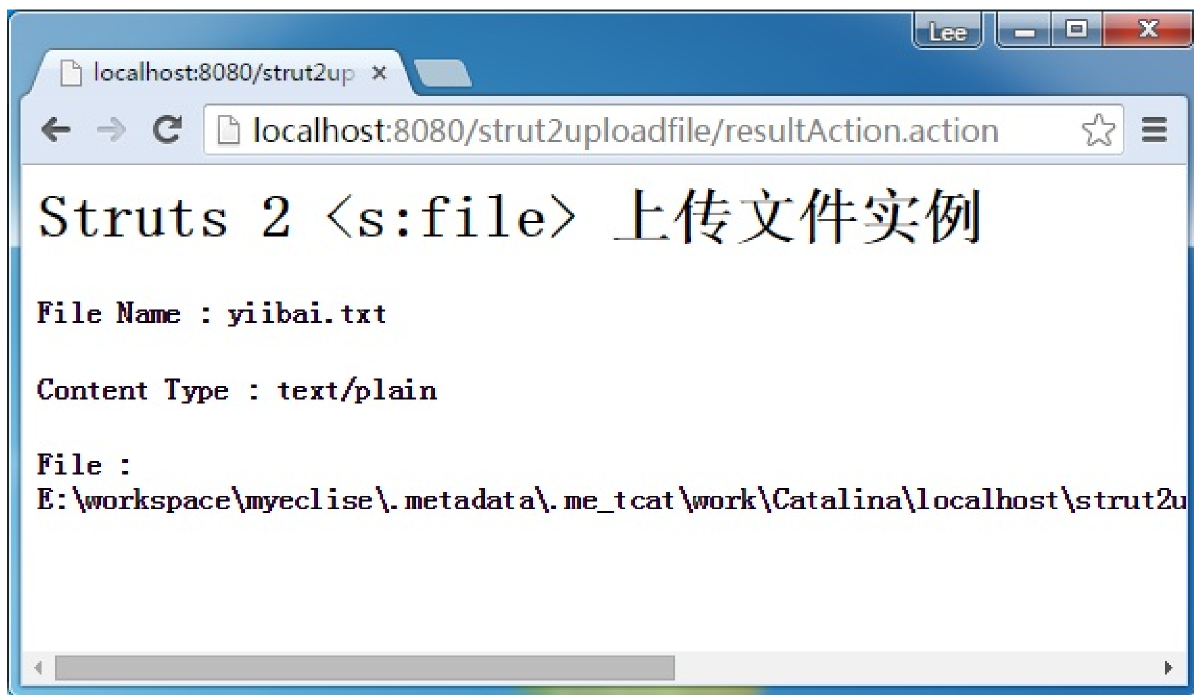
*http://localhost:8080/Struts2Example/fileUploadAction.action*




错误信息提示，如果您上传一个文件，该文件超过10KB，或者未选文本文件。




上传名为“yiibai.com.txt”的文本文件, 文件大小：5kb.




上传的文件将被视为一个临时文件，具有长的随机文件名，如：upload376584a7_12981122379 8000_00000010.tmp. 请确保这个临时文件复制到其他地方。 阅读文件实用文档复制文件。

## 参考

1. Struts 2 文件文档
2. http://struts.apache.org/2.0.14/docs/file-upload.html
3. http://struts.apache.org/2.0.14/docs/how-do-we-upload-files.html
4. http://commons.apache.org/io/api-1.4/org/apache/commons/io/FileUtils.html

下载代码 – http://pan.baidu.com/s/1eQDH07S

# Struts2资源包使用示例 - Struts2教程

要使用资源包从属性文件检索消息，必须了解Struts2的资源包搜索顺序：

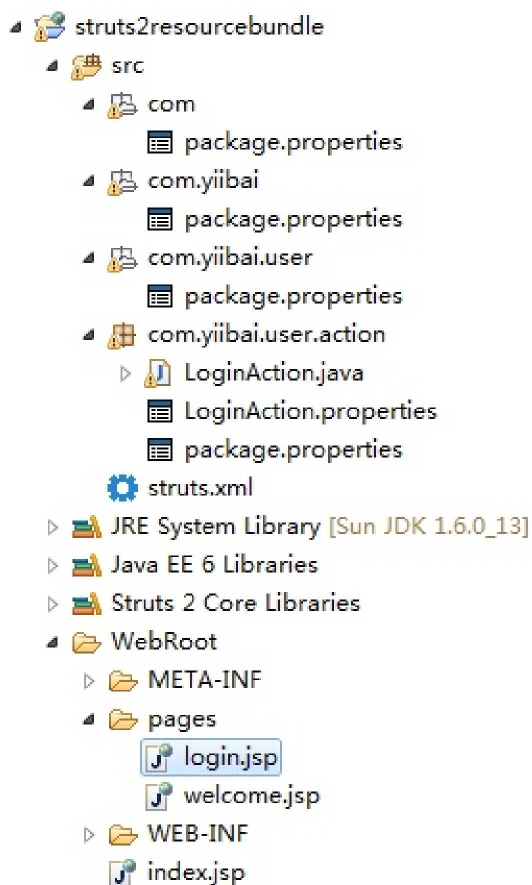
## 资源包搜索顺序

资源包中搜索按以下顺序：

1. ActionClass.properties
2. Interface.properties
3. BaseClass.properties
4. ModelDriven's model
5. package.properties
6. 搜索国际化消息键的层次结构本身
7. 全局资源属性

请参阅Struts2资源包文档详细解释。

在实践中，是不可能组织属性的文件的顺序。所以，只要了解几个常用的搜索顺序应该是足够了： package.properties 和 global resource properties。参见下图：




如果com.yiibai.user.action.LoginAction想通过资源包获得消息，它将搜索

1. com.yiibai.user.action.LoginAction.properties (找到，退出，否则下一个)
2. com.yiibai.user.action.package.properties (找到，退出，否则下一个)
3. com.yiibai.user.package.properties (找到，退出，否则下一个) …一路不断在每个父目录的根目录查找package.properties
4. 查找全局资源属性，如果将其配置在应用程序中。

明白这搜索顺序可以给你更多的信心来决定正确的文件夹的属性文件。

## 获取资源包

下面是访问该资源包的几个例子：

P.S ‘username.required‘ 和 ‘username‘ 在一个属性文件中的键。

## 1. 动作类

在Action类，可以扩展了ActionSupport和通过getText(‘key’) 函数获取资源包。

```
...
public class LoginAction extends ActionSupport{
    ...
    public void validate(){
        if("".equals(getUsername())){
            addFieldError("username", getText("username.required")
        }
    }
}
```

## 2. <s:property>标签

在属性标记，使用 getText(‘key’).

```
<s:property value="getText('username')" />
```

## 3. <s:text>标签

在text标签，设置“name”属性的键。

```
<s:text name="username" />
```

## 4. Key属性

UI组件的主要属性有特殊的功能，查看这个key属性例子详细信息。

```
<s:textfield key="username" />
```

## 5. I18n标签

国际化i18n 标签可以从“name”属性声明指定资源包得到消息。在这个例子中，它要求从com/yiibai/user/package.properties文件中以获得 “username” 的消息。

```
<s:i18n name="com.yiibai.user.package" >
     <s:text name="username" />
</s:i18n>
```

访问 URL http://localhost:8080/struts2resourcebundle/user/login.action，输出以下结果：



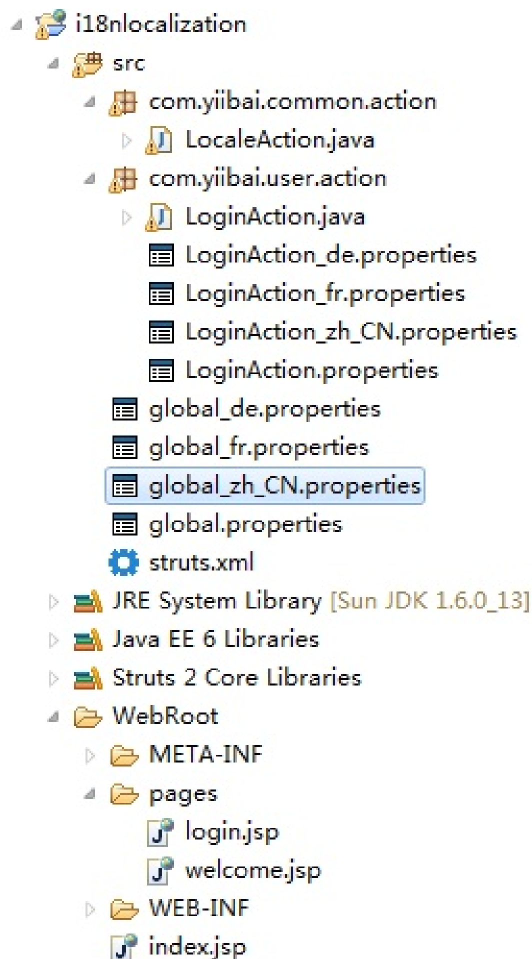

下载完整的项目实践(struts2resourcebundle) – http://pan.baidu.com/s/1dD2UQ2I

# Struts2本地化和国际化 - Struts2教程

Struts 2的国际化(I18N)和本地化(i10n)或多语言的例子，来说明如何使用资源包来显示不同语言的消息。在这个例子中，您将创建一个简单的登录屏幕，通过Struts 2的UI组件显示来自资源包的消息， 并更改基于所选的语言选项的语言环境。

## 1. 工程结构

项目结构，如下图片显示：




## 2. Properties文件

确保属性文件命名为国家指定的代码。在一些“非欧洲”或“非英语”之类的字符，应该始终编码的内容 native2ascii属性

**global.properties**

```
#Global messages
global.username = Username
global.password = Password
global.submit = Submit
```

### global_zh_CN.properties

```
#Global messages
global.username = \u7528\u6237\u540d
global.password = \u5bc6\u7801
global.submit=\u63d0\u4ea4
```

### global_fr.properties

```
#Global messages
global.username = Nom d'utilisateur
global.password = Mot de passe
global.submit = Soumettre
```

### global_de.properties

```
#Global messages
global.username = Benutzername
global.password = Kennwort
global.submit = Einreichen
```

请仔细阅读Struts2资源包的例子来了解Struts 2的自动搜索属性文件。

## 3. 动作类

两个动作类，LocaleAction基本上是什么都不做， 而 LoginAction 会做一个简单的验证和通过gettext()显示来自资源包错误信息。

LocaleAction.java

```
package com.yiibai.common.action;

import com.opensymphony.xwork2.ActionSupport;

public class LocaleAction extends ActionSupport{

    //business logic
    public String execute() {
        return "SUCCESS";
    }
}
```

**LoginAction.java**

```
package com.yiibai.user.action;

import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport{

    private String username;
    private String password;

    //...getter and setter methods

    //business logic
    public String execute() {
        return "SUCCESS";
    }

    //simple validation
    public void validate(){
        if("".equals(getUsername())){
            addFieldError("username", getText("username.required")
        }
        if("".equals(getPassword())){
            addFieldError("password", getText("password.required")
        }
    }
}
```

# 4. 视图页面

一个登录页面带有一个文本框，密码和提交UI组件。为了支持Struts2 本地化，必须声明 <%@ page contentType="text/html;charset=UTF-8" %>在您的视图页面，否则能将有问题的"UTF-8数据"正确显示，尤其是中国汉字。阅读这篇文章，关于Struts2中国本土化问题。

**login.jsp**

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>

<body>
<h1>Struts 2 localization example</h1>

<s:form action="validateUser" namespace="/user">

    <s:textfield key="global.username" name="username" />
    <s:password key="global.password" name="password"/>
    <s:submit key="global.submit" name="submit" />

</s:form>

<s:url id="localeEN" namespace="/" action="locale" >
   <s:param name="request_locale" >en</s:param>
</s:url>
<s:url id="localezhCN" namespace="/" action="locale" >
   <s:param name="request_locale" >zh_CN</s:param>
</s:url>
<s:url id="localeDE" namespace="/" action="locale" >
   <s:param name="request_locale" >de</s:param>
</s:url>
<s:url id="localeFR" namespace="/" action="locale" >
   <s:param name="request_locale" >fr</s:param>
</s:url>

<s:a href="%{localeEN}" >English</s:a>
<s:a href="%{localezhCN}" >Chinese</s:a>
<s:a href="%{localeDE}" >German</s:a>
<s:a href="%{localeFR}" >France</s:a>

</body>
</html>
```

要更改默认的语言环境，只需要声明“request_locale”参数，设置你喜欢的语言代码，并传递给一个Action类。在 Struts2中，com.opensymphony.xwork2.interceptor.I18nInterceptor 拦截器, 在 struts-default.xml中声明将拦截Action类，并相应地处理语言环境。

# 5. 显示资源包的消息？

在Struts2，有很多的方式来显示所选择的语言或语言环境的资源包的信息。有关示例，

```
<s:textfield key="global.username" name="username" />
<s:text name="global.username" />
<s:property value="getText('global.username')" />
<s:text name="global.password" />
```

在Struts1，有一个标准的 bean:message 来显示资源包的消息。但是在Struts 2 中，有这么多相当于显示资源包的消息(甚至内部的工作不同)方式，基本上，无论选择的是什么，在 Struts2 也将显示正确的资源包的消息。

# 6. struts.xml

Struts2 的配置文件，链接一起。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

    <constant name="struts.custom.i18n.resources" value="global" /
     <constant name="struts.devMode" value="true" />

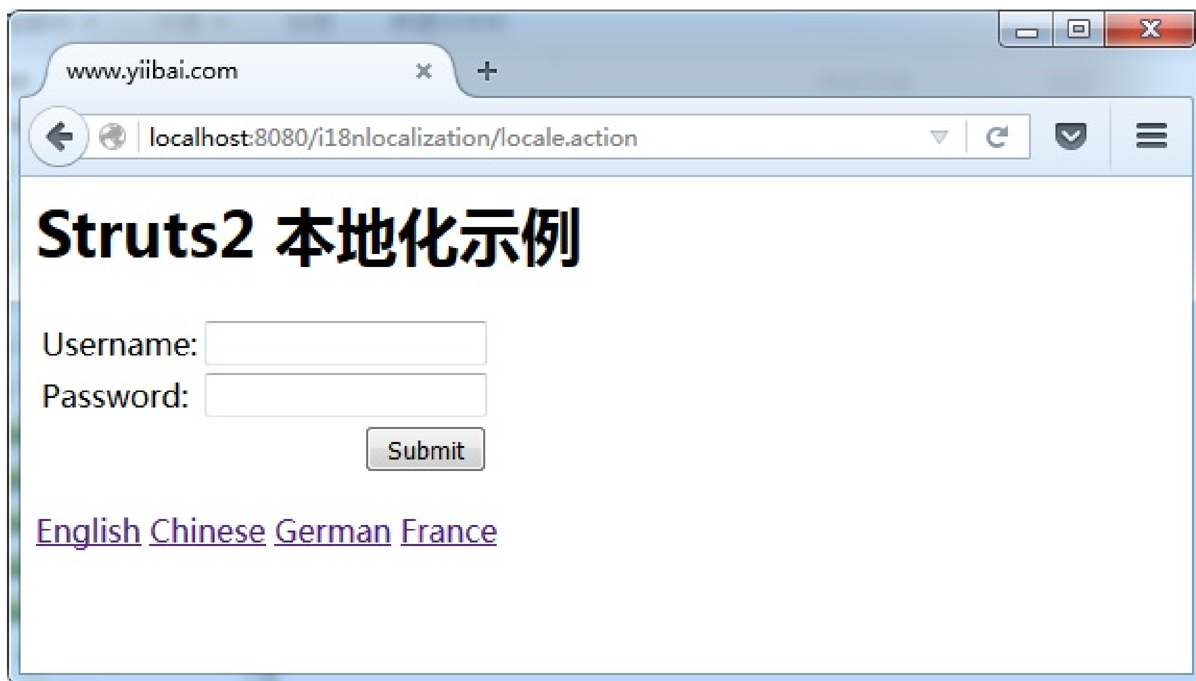
    <package name="user" namespace="/user" extends="struts-default'
       <action name="login">
        <result>/pages/login.jsp</result>
       </action>
       <action name="validateUser" class="com.yiibai.user.action.Lo
        <result name="SUCCESS">/pages/welcome.jsp</result>
        <result name="input">/pages/login.jsp</result>
       </action>
    </package>

    <package name="default" namespace="/" extends="struts-default">
       <action name="locale" class="com.yiibai.common.action.Locale
        <result name="SUCCESS">/user/pages/login.jsp</result>
       </action>
    </package>

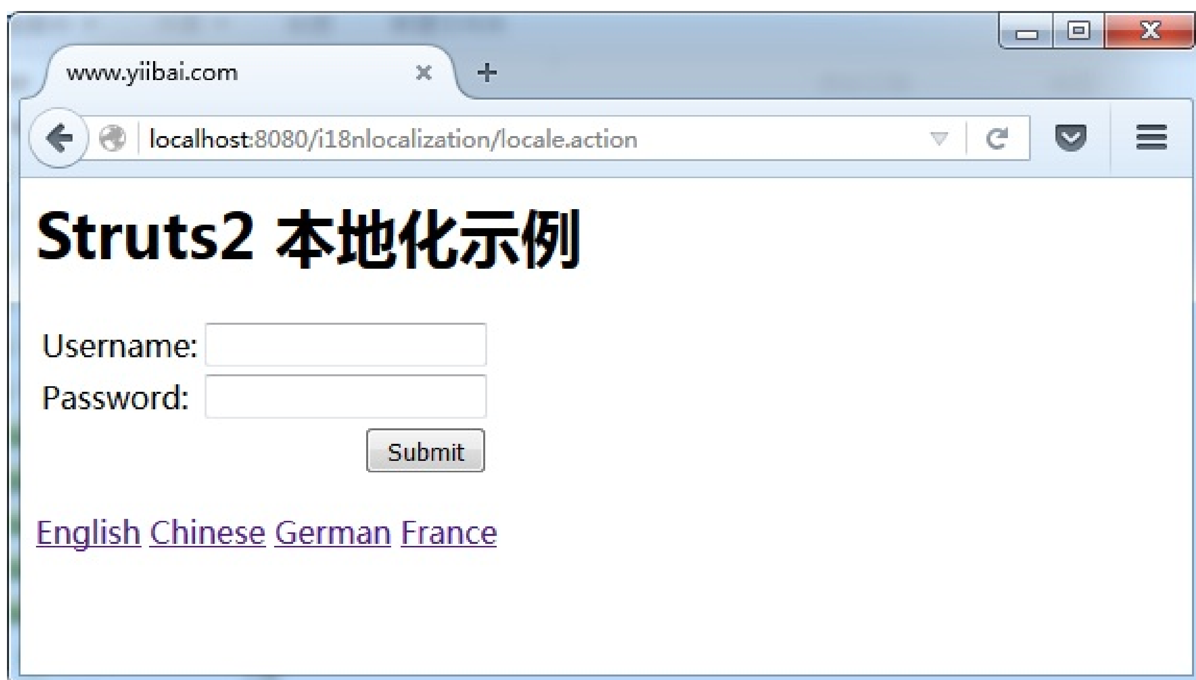
</struts>
```

# 7. 示例

http://localhost:8080/i18nlocalization/user/login.action




http://localhost:8080/i18nlocalization/locale.action?request_locale=en



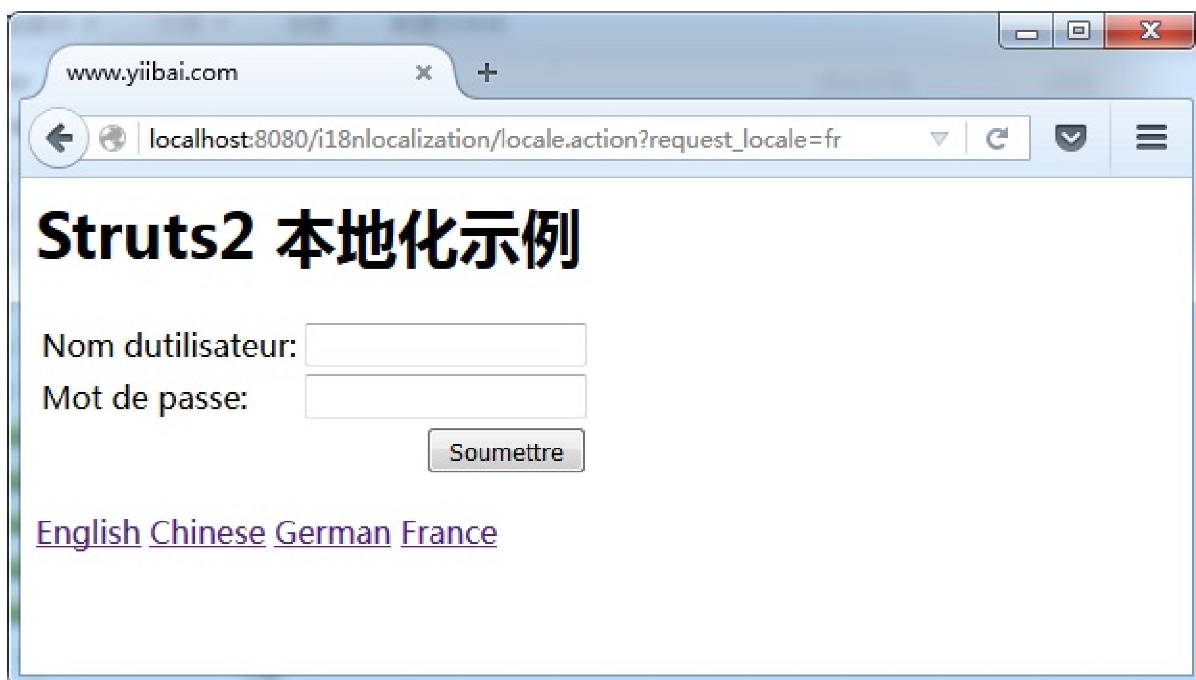

http://localhost:8080/i18nlocalization/locale.action?request_locale=zh_CN



http://localhost:8080/i18nlocalization/locale.action?request_locale=de




http://localhost:8080/i18nlocalization/locale.action?request_locale=fr




## 参考


1. http://struts.apache.org/2.1.8/docs/localization.html
2. http://www.yiibai.com/java/java-convert-chinese-character-to-unicode-with-native2ascii/
3. http://www.yiibai.com/struts2/struts-2-resource-bundle-example/
4. http://www.yiibai.com/struts/struts-internationalizing-or-localization-example/


代码下载 - http://pan.baidu.com/s/1jGCUaJ8

# Struts2 key键属性示例 - Struts2教程

在Struts2，在UI组件的“key”属性来处理本地化的常用方法，也是编码UI标签的一个非常有效的方式。见下面两种情况：

## 1. Properties属性文件

属性文件包含一条消息。

**global.properties**

```
global.username = Username
```

## 2. 示例1

如果分配一个“key”属性到一个文本框。键(key)属性会从资源包中获取信息，并使其在默认XHTML text.tfl模板基础上渲染。

```
<s:form action="validateUser">
    <s:textfield key="global.username" />
</s:form>
```

现在它将会使用“global.username {left-side}”和 “Username {right-side}”，并匹配相应的XHTML text.tfl模板。

```
<td class="tdLabel">
   <label for="validateUser_{left-side}" class="label">{right-side]
</td>
<td>
   <input type="text" name="{left-side}" value="" id="validateUser_
</td>
```

最后的 HTML

```
<td class="tdLabel">
   <label for="validateUser_global_username" class="label">Usernam
</td>
<td>
   <input type="text" name="global.username" value="" id="validateU
</td>
```

键属性将使用 {left-side}作为文本框名称和ID; {right-side} 作为标签值。

# 3. 示例2

在某些情况下，可能需要显式声明的一个不同的名称的文本框。

```
<s:form action="validateUser">
    <s:textfield key="global.username" name="username"/>
</s:form>
```

现在key属性将使用“Username {right-side}”来只匹配的标签值， 文本框的名称和ID将明确覆盖。

最后的 HTML

```
<td class="tdLabel">
   <label for="validateUser_username" class="label">Username:</labe
</td>
<td>
   <input type="text" name="username" value="" id="validateUser_use
</td>
```

key属性可以提高你的开发速度，使代码更有效，这是值得学习的。

# Struts2中文本地化问题 - Struts2教程

一个 Struts2 的国际化定位的问题，用来显示中国汉字...

## 案例1：属性有特殊字符的文件

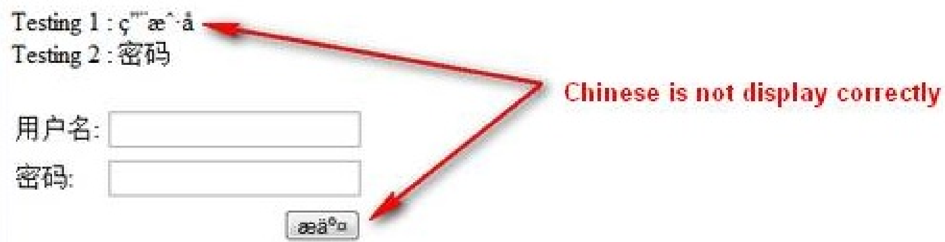
属性文件存储用户名，密码信息，并以中文字符提交。此属性文件以UTF-8格式创建的，但内容不使用 native2ascii 编码。




让我们试着通过一些UI标签，来显示中国汉字。查看页面声明为UTF-8格式的HTML元标记来显示。

```
...
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
</head>
...
<s:form action="validateUser">
    <s:textfield key="global.username" name="username"/>
    <s:password key="global.password" name="password"/>
    <s:submit key="global.submit" name="submit" />

    <div>Testing 1 : <s:property value="getText('global.username')'
    <div>Testing 2 : <s:text name="global.password" /></div></br/>
</s:form>
...
<s:url id="localezhCN" namespace="/" action="locale" >
   <s:param name="request_locale" >zh_CN</s:param>
</s:url>
...
<s:a href="%{localezhCN}" >Chinese</s:a>
...
```




结果

令人惊奇的是，以下三个UI标签都能够正确地显示中国消息

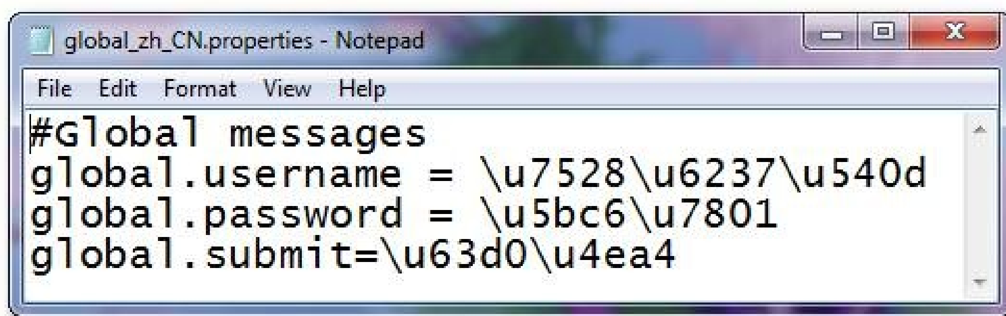
```
<s:textfield key="global.username" name="username"/>
<s:password key="global.password" name="password"/>
Testing 2 : <s:text name="global.password" />
```

然而，“s:submit” 和 “getText()” 却无法显示呢？据Java的国际化文档，要使用资源包正确显示特殊字符，则必须用 native2ascii 工具进行处理。

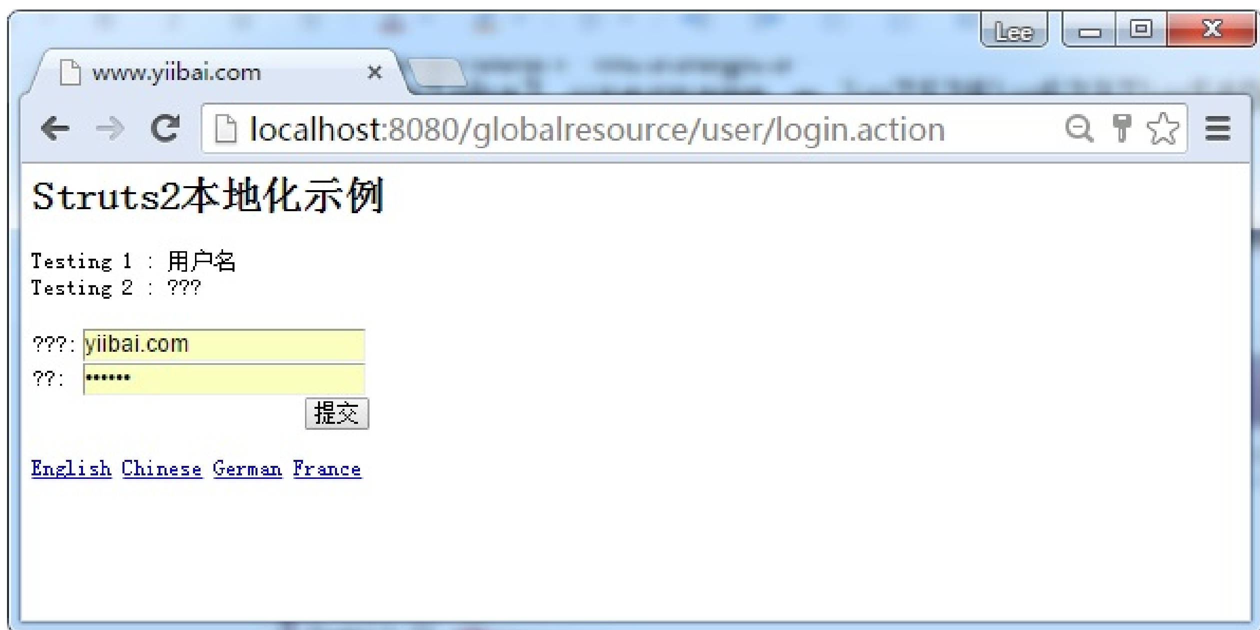
深入到 TextProvider.getText()的源代码后，它使用的资源 bundle.getString()来从资源包检索的消息，所以不正确的消息是合理的。但是，为什么“s:text“, “s:textfield” 和 “s:password” 能够正确显示了中文的消息，为什么“s:submit”会失败？在有太多的问题，让我们看看示例2...

# 案例2：有特殊字符的属性文件(编码)

这一次，属性文件使用native2ascii工具处理中国汉字的编码正确。




结果如下所示：




其结果是完全逆转，现在 “s:submit” 和 “getText()” 是能够正确地显示它，但其他UI组件失败。 这里是按预期方式工作的，因为在Struts 2推荐使用getText()，以显示国际化或本地化的消息。问题是，为什么“s:submit”会不同呢？

## Struts2..哪里有问题？

这里有几个问题：

1. 为什么 s:submit 有如此不同的效果？
2. 对国际化应该是非常简单的，为什么在Struts 2有这种问题？ 或者我们误解了Struts2 国际化如何工作了？
3. 为什么有这么多的方式来显示来自资源包的消息？为什么不直接组织成一个方法？ 在Struts1，只需使用“bean:message”，为什么Struts 2中它看起来很复杂？
4. Struts2的支持XML资源包？ 我们可能不太喜欢用native2ascii工具对数据进行编码为UTF-8格式，它使属性文件不可读。 Apache Wicket的在这个问题做了很好的工作，可能是在Struts 2中吸取教训。
5. 那么，如何正确地在 Struts2 中显示中国汉字？

许多文章和教程使用以下方法来显示资源包的消息：

```
<s:text name="global.username"/>
<s:property value="getText('global.username')"/>
```

然而，这仅适用于英国或一些“英语状(欧洲)” 的字符，如法文，德文。但对中文或日文，这两种方法将返回完全不同的输出。真的不知道Struts2的本地化该怎么办了。

## 解决办法

问题是在HTML meta标签，

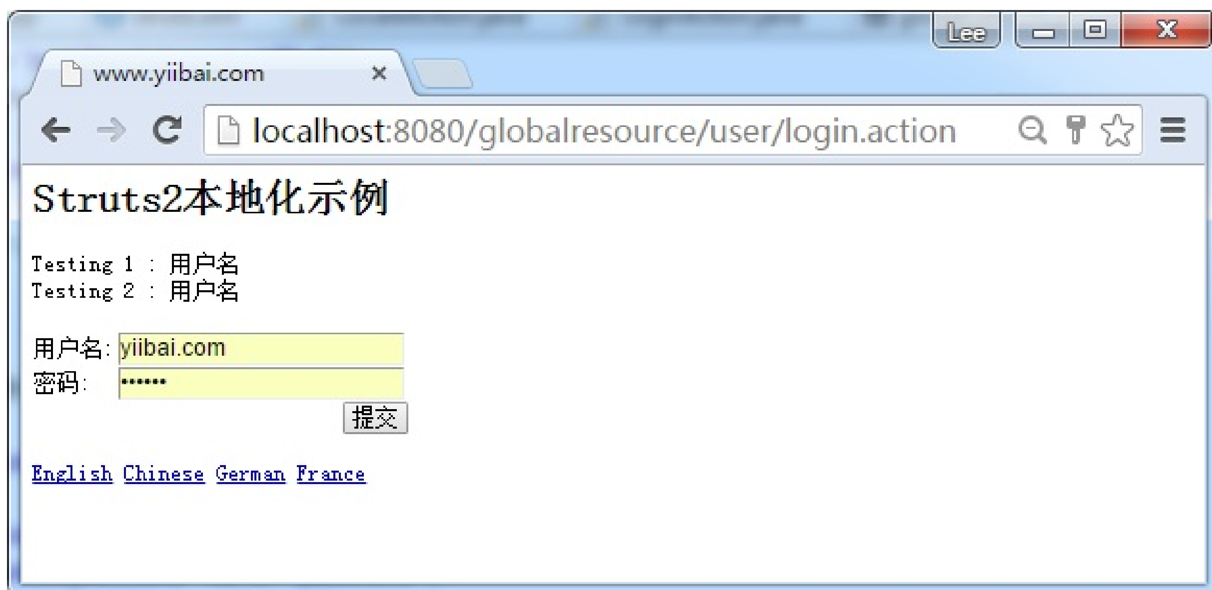
```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
</head>
```

在Struts1，上述meta标签必须正确显示UTF-8的数据，但这在 Struts2 是有问题的。

在Struts2，meta标签不起作用，我们应该把 <%@ page contentType="text/html;charset=UTF-8" %>标签放在页面的第一行。例如下面的代码片断：

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>
...
```

结果显示如下：




所有的中文消息正确显示。

# 回答之前的问题

1. 为什么 s:submit 有如此不同的效果？**A:** 暂无评论
2. 对国际化应该是非常简单的，为什么在Struts 2有这种问题？ 或者我们误解了Struts2 国际化如何工作了？**A:** 确保把** <%@ page contentType="text/html;charset=UTF-8" %>" 放在页面的第一行。**

3. 为什么有这么多的方式来显示来自资源包的消息？为什么不直接组织成一个方法？ 在Struts1，只需使用“bean:message”，为什么Struts 2中它看起来很复杂？**A: s:text, key, getText(), name… ,** 所有的都能够正确地显示中文或UTF-8编码的数据，只要确保把正确的“字符集”放在视图页面中。 **
4. Struts2的支持XML资源包？ 我们可能不太喜欢用native2ascii工具对数据进行编码为UTF-8格式，它使属性文件不可读。 Apache Wicket的在这个问题做了很好的工作，可能是在Struts 2中吸取教训。**A:**希望在Struts2的下一版本可以支持在XML资源包。
5. 那么，如何正确地在 Struts2 中显示中国汉字？**A:** 看看上页的解决办法

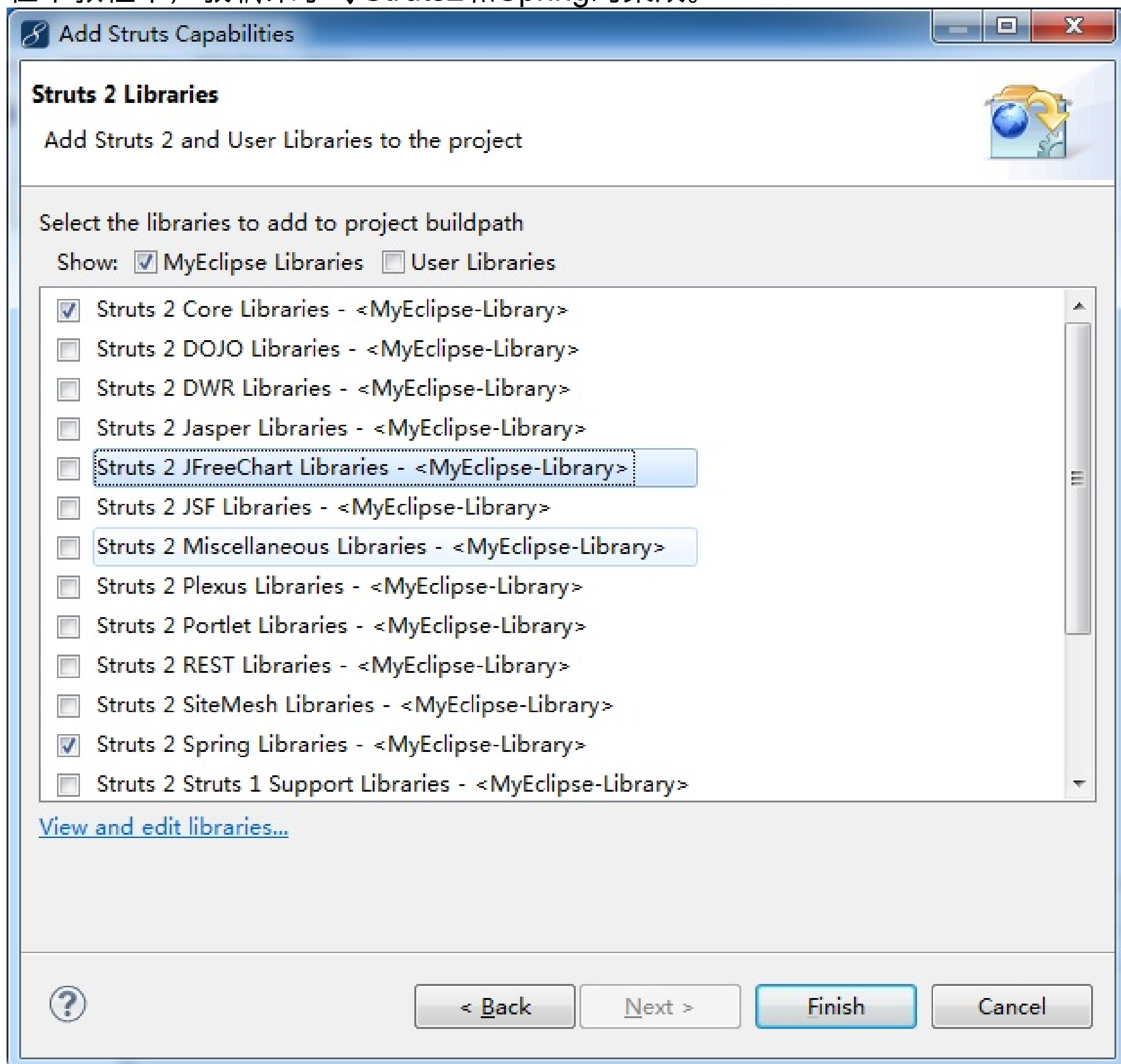
# 参考


1. http://www.yiibai.com/java/java-convert-chinese-character-to-unicode-with-native2ascii.html
2. http://forums.sun.com/thread.jspa?threadID=5185040
3. http://www.coderanch.com/t/452139/Struts/applicationresources-properties-utf-characters#2013557
4. http://struts.apache.org/2.1.8/docs/localization.html
5. http://hxzon00.blog.163.com/blog/static/10489241620088121449163/


下载代码(globalresource) – http://pan.baidu.com/s/1mgzt3dQ

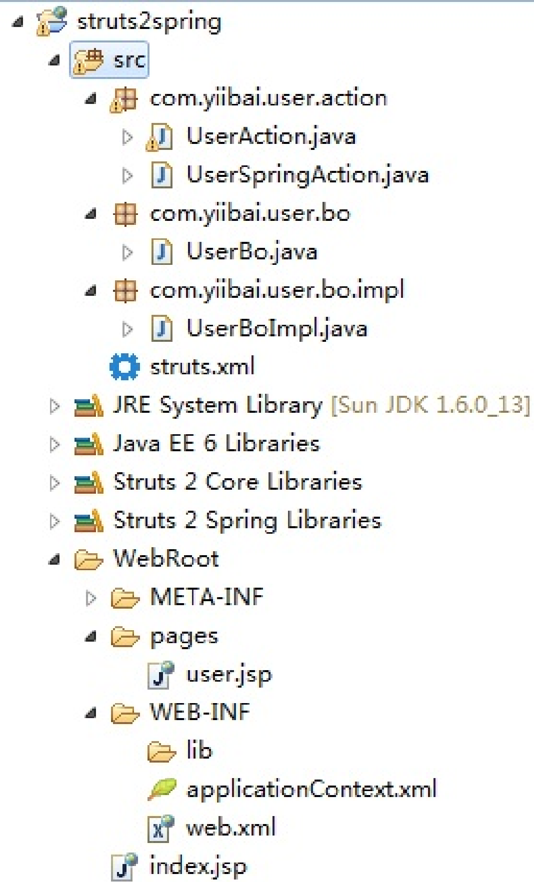
# Struts2+Spring集成实例 - Struts2教程

在本教程中，我们来学习Struts2和Spring的集成。




## 1. 工程结构

下面的图是本教程的项目文件夹结构。




## 2. Spring监听器

配置Spring监听器 “org.springframework.web.context.ContextLoaderListener” 到 web.xml 文件中。

**web.xml**

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Struts 2 Web Application</display-name>

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
         org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndE
        </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

</web-app>
```

## 3. 注册Spring Bean

注册所有的Spring Beans 配置在 applicationContext.xml 文件中, Spring监听器会自动找到这个 XML 文件。

**applicationContext.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="userBo" class="com.yiibai.user.bo.impl.UserBoImpl" /

    <bean id="userSpringAction" class="com.yiibai.user.action.UserS
        <property name="userBo" ref="userBo" />
    </bean>

</beans>
```

### UserBo.java

```
package com.yiibai.user.bo;

public interface UserBo{

    public void printUser();

}
```

### UserBoImpl.java

```
package com.yiibai.user.bo.impl;

import com.yiibai.user.bo.UserBo;

public class UserBoImpl implements UserBo{

    public void printUser(){
        System.out.println("printUser() is executed...");
    }

}
```

### UserSpringAction.java

```
package com.yiibai.user.action;

import com.yiibai.user.bo.UserBo;

public class UserSpringAction{

    //DI via Spring
    UserBo userBo;

    public UserBo getUserBo() {
        return userBo;
    }

    public void setUserBo(UserBo userBo) {
        this.userBo = userBo;
    }

    public String execute() throws Exception {

        userBo.printUser();
        return "success";

    }
}
```

# 4. Struts.xml

在此声明的所有关系。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
     <constant name="struts.devMode" value="true" />

    <package name="default" namespace="/" extends="struts-default">

        <action name="userAction"
            class="com.yiibai.user.action.UserAction" >
            <result name="success">pages/user.jsp</result>
        </action>

        <action name="userSpringAction"
            class="userSpringAction" >
            <result name="success">pages/user.jsp</result>
        </action>

    </package>

</struts>
```

## 5. 示例

现在，所有的Struts2和Spring的集成工作已经完成，现在看到下面的用例来访问Spring 的 “userBo” Bean。

- 用例 1 : 让 Spring 充当 Struts2的Action类，并访问Spring的Bean。
- 用例 2 : 在Struts2的Action类中访问Spring的Bean。

## 用例1

在这个例子中，userSpringAction充当Struts2的Action类，也可以使用普通Spring的方式注入Spring的userBo。

```
//struts.xml
<action name="userSpringAction"
    class="userSpringAction" >
    <result name="success">pages/user.jsp</result>
</action>

//applicationContext.xml
<bean id="userSpringAction" class="com.yiibai.user.action.UserSprin
    <property name="userBo" ref="userBo" />
</bean>
```

要访问此操作，请使用网址：
http://localhost:8080/struts2spring/userSpringAction.action

# 用例 2

默认情况下，Spring监听器启用 “通过匹配bean的名字自动装配“。 因此，它会通过setUserBo自动传递Spring “userBo” Bean 到UserAction。请参阅下面的Struts2动作：

Spring的自动装配功能可以修改为 **name**(默认), **type**, **auto** 或 **constructor**, 可能需要参考 Struts2的Spring插件文档。

**UserAction.java**

```
package com.yiibai.user.action;

import com.yiibai.user.bo.UserBo;
import com.opensymphony.xwork2.ActionSupport;

public class UserAction extends ActionSupport{

    //DI via Spring
    UserBo userBo;

    public UserBo getUserBo() {
        return userBo;
    }

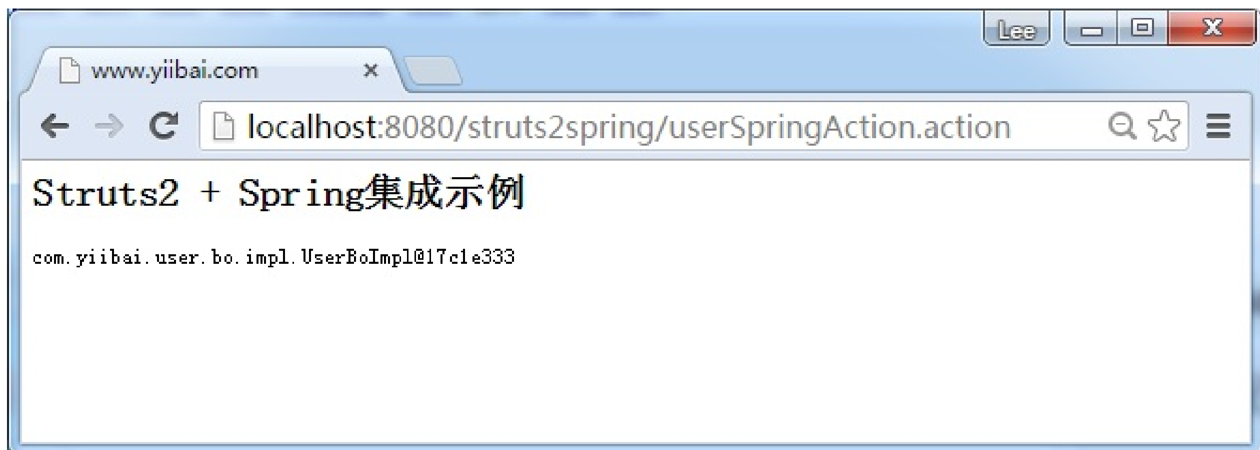
    public void setUserBo(UserBo userBo) {
        this.userBo = userBo;
    }

    public String execute() throws Exception {

        userBo.printUser();
        return SUCCESS;

    }
}
```

要访问此操作，请使用网址： http://localhost:8080/struts2spring/userAction.action




**WebApplicationContextUtils** 另外，也可以使用Spring 通用 **WebApplicationContextUtils** 类来直接获得Spring的bean。

```
package com.yiibai.user.action;

import org.apache.struts2.ServletActionContext;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContex

import com.yiibai.user.bo.UserBo;
import com.opensymphony.xwork2.ActionSupport;

public class UserAction extends ActionSupport{

    public String execute() throws Exception {

        WebApplicationContext context =
            WebApplicationContextUtils.getRequiredWebApplicationCon
                                    ServletActionContext.getServlet
                        );

        UserBo userBo1 = (UserBo)context.getBean("userBo");
        userBo1.printUser();

        return SUCCESS;

    }
}
```

这是一个又长又臭的文章(包教不包会)，请下载完整的项目并按照源代码去一步步实现。

# 参考

1. Struts2 Spring 插件文档
2. Struts + Spring 集成实例

代码下载 - http://pan.baidu.com/s/1dDhqQ5b

# Struts2+Hibernate使用Full Hibernate Plugin集成 - Struts2教程

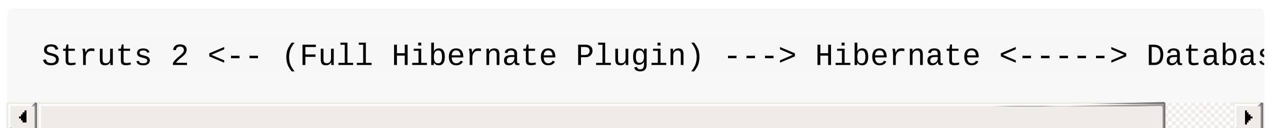
在上篇 Struts2 + Hibernate集成 实例中, 它使用 servlet 上下文监听 Hibernate 的 Session，而且把Struts2和Hibernate框架集成。

但是，总有一些东西要提高。在本教程中，我们将展示如何整合 Struts2+Hibernate，并使用Struts2一个名为“Full Hibernate Plugin“的插件。

见下面的集成步骤：

1. 把 “Full Hibernate Plugin” jar 放入到工程类路径。
2. 使用 “@SessionTarget” 注释来注入到 Hibernate session; 当“@TransactionTarget” 注释注入到Hibernate 事务。
3. 在 struts.xml, 让包扩展 “hibernate-default“，而不是默认的堆栈。

看看下面的关系：

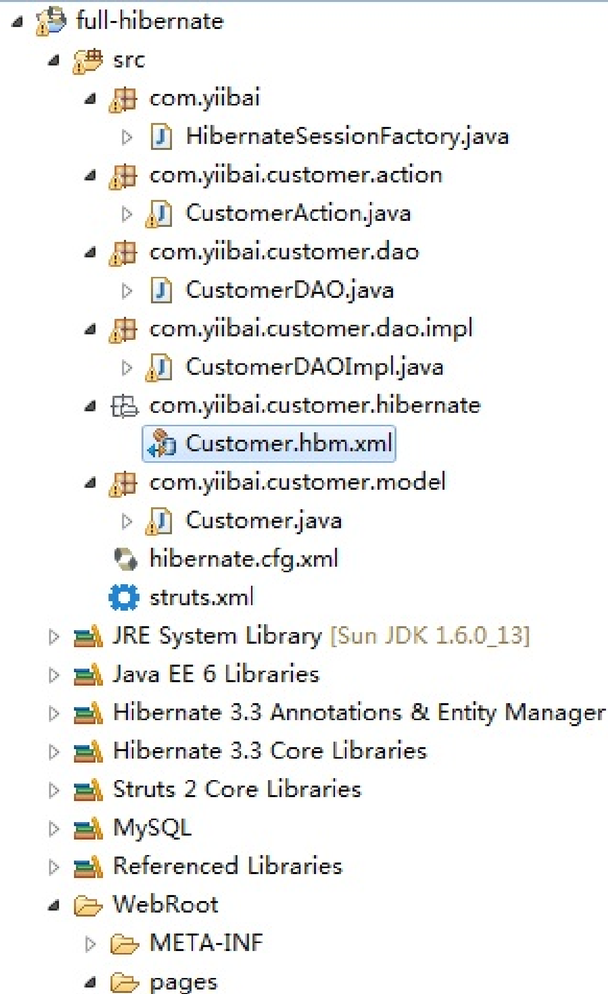


```
Struts 2 <-- (Full Hibernate Plugin) ---> Hibernate <-----> Databas
```

注， 本教程是从以前的 Struts2 + Hibernate集成 实例(servlet context listener)更新版本。因此，JSP 和 Hibernate 配置基本相同，只是整合的部分是有点不同，尝试比较既能发现不同。

## 1. 工程结构

在节教程，我们创建一个工程名为 full-hibernate 的web工程。看看这个项目文件夹




的完整结构。

## 2. MySQL创建表脚本

Customer表结构

```
CREATE TABLE `customer` (
  `customer_id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `address` varchar(255) NOT NULL,
  `create_date` datetime NOT NULL,
  PRIMARY KEY (`customer_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

## 3. Hibernate相关配置

所有 Hibernate 的模型和配置的东西。

**Customer.java** – 为customer 表创建一个类。

```
package com.yiibai.customer.model;

import java.util.Date;

public class Customer implements java.io.Serializable {

    private Long customerId;
    private String name;
    private String address;
    private Date createdDate;

    //getter and setter methods
}
```

**Customer.hbm.xml** – Hibernate 的 customer 表映射。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 20 Julai 2010 11:40:18 AM by Hibernate Tools 3.2.5.E
<hibernate-mapping>
    <class name="com.yiibai.customer.model.Customer"
        table="customer" catalog="yiibai">
        <id name="customerId" type="java.lang.Long">
            <column name="CUSTOMER_ID" />
            <generator class="identity" />
        </id>
        <property name="name" type="string">
            <column name="NAME" length="45" not-null="true" />
        </property>
        <property name="address" type="string">
            <column name="ADDRESS" not-null="true" />
        </property>
        <property name="createdDate" type="timestamp">
            <column name="CREATED_DATE" length="19" not-null="true'
        </property>
    </class>
</hibernate-mapping>
```

文件: hibernate.cfg.xml, Hibernate 数据库配置

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.bytecode.use_reflection_optimizer">fa
    <property name="hibernate.connection.password">password</proper
    <property name="hibernate.connection.url">jdbc:mysql://localhos
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLD
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="use_sql_comments">false</property>
    <mapping resource="com/yiibai/customer/hibernate/Customer.hbm.x
  </session-factory>
</hibernate-configuration>
```

# 5. DAO

实现DAO设计模式执行数据库操作。在 CustomerDAOImpl 类, 声明Hibernate会话和事务为类成员。在Struts 2的项目初始化，"Full Hibernate Plugin" 使用 @SessionTarget 和 @TransactionTarget 分别标注将注入相应的 Hibernate 会话和事务成为类成员。

### CustomerDAO.java

```
package com.yiibai.customer.dao;

import java.util.List;

import com.yiibai.customer.model.Customer;

public interface CustomerDAO{

    void addCustomer(Customer customer);

    List<Customer> listCustomer();

}
```

### CustomerDAOImpl.java

```
package com.yiibai.customer.dao.impl;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.Transaction;

import com.googlecode.s2hibernate.struts2.plugin.annotations.Sessic
import com.googlecode.s2hibernate.struts2.plugin.annotations.Transa
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.model.Customer;

public class CustomerDAOImpl implements CustomerDAO{

    @SessionTarget
    Session session;

    @TransactionTarget
    Transaction transaction;

    //add the customer
    public void addCustomer(Customer customer){

        session.save(customer);

    }

    //return all the customers in list
    public List<Customer> listCustomer(){

        return session.createQuery("from Customer").list();

    }

}
```

# 6. Action

在Action类，调用DAO类来执行数据库操作。

**CustomerAction.java**

```
package com.yiibai.customer.action;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
```

```
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.dao.impl.CustomerDAOImpl;
import com.yiibai.customer.model.Customer;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

public class CustomerAction extends ActionSupport
    implements ModelDriven{

    Customer customer = new Customer();
    List<Customer> customerList = new ArrayList<Customer>();
    CustomerDAO customerDAO = new CustomerDAOImpl();

    public String execute() throws Exception {
        return SUCCESS;
    }

    public Object getModel() {
        return customer;
    }

    public List<Customer> getCustomerList() {
        return customerList;
    }

    public void setCustomerList(List<Customer> customerList) {
        this.customerList = customerList;
    }

    //save customer
    public String addCustomer() throws Exception{

        //save it
        customer.setCreatedDate(new Date());
        customerDAO.addCustomer(customer);

        //reload the customer list
        customerList = null;
        customerList = customerDAO.listCustomer();

        return SUCCESS;

    }

    //list all customers
    public String listCustomer() throws Exception{

        customerList = customerDAO.listCustomer();

        return SUCCESS;

    }
```

```
}
```

# 7. JSP 页面

JSP页面添加并列出客户。

**customer.jsp**

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>

<body>
<h1>Struts 2 Full Hibernate Plugin example</h1>

<h2>Add Customer</h2>
<s:form action="addCustomerAction" >
  <s:textfield name="name" label="Name" value="" />
  <s:textarea name="address" label="Address" value="" cols="50" row
  <s:submit />
</s:form>

<h2>All Customers</h2>

<s:if test="customerList.size() > 0">
<table border="1px" cellpadding="8px">
    <tr>
        <th>Customer Id</th>
        <th>Name</th>
        <th>Address</th>
        <th>Created Date</th>
    </tr>
    <s:iterator value="customerList" status="userStatus">
        <tr>
            <td><s:property value="customerId" /></td>
            <td><s:property value="name" /></td>
            <td><s:property value="address" /></td>
            <td><s:date name="createdDate" format="dd/MM/yyyy" /></
        </tr>
    </s:iterator>
</table>
</s:if>
<br/>
<br/>

</body>
</html>
```

# 8. struts.xml

链接所有～让包扩展 “hibernate-default” 来代替 “struts-default“.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
  <constant name="struts.devMode" value="true" />

  <package name="default" namespace="/" extends="hibernate-default'

    <action name="addCustomerAction"
    class="com.yiibai.customer.action.CustomerAction" method="addCu
       <result name="success">pages/customer.jsp</result>
    </action>

    <action name="listCustomerAction"
    class="com.yiibai.customer.action.CustomerAction" method="list(
        <result name="success">pages/customer.jsp</result>
    </action>

</package>
</struts>
```

# 9. 实例

访问以下网址：http://localhost:8080/full-hibernate/addCustomerAction.action

pc-20151109psmq:8080/full-hibernate/addCustomerAction.action

# Struts2 Full Hibernate Plugin 实例

## 添加顾客

Name:

Address:

Submit

## 所有顾客

| Customer Id | Name | Address | Created Date |
|---|---|---|---|
| 17 | yiibai | Hainan HaiKou | 02/12/2015 |

pc-20151109psmq:8080/full-hibernate/addCustomerAction.action

# Struts2 Full Hibernate Plugin 实例

## 添加顾客

Name:

Address:

Submit

## 所有顾客

| Customer Id | Name | Address | Created Date |
|---|---|---|---|
| 17 | yiibai | Hainan HaiKou | 02/12/2015 |
| 18 | 易百教程 | 海南海口 | 02/12/2015 |

## 参考

1. Struts2 Full Hibernate插件文档
2. Struts2 + Hibernate集成实例
3. 安装库到Maven本地资源库

## 下载代码

下载所有源代码 – http://pan.baidu.com/s/1o6tjSam

# Struts2+Hibernate集成实例 - Struts2教程

在 Struts2 中，没有官方的插件集成Hibernate框架。但是，可以通过以下步骤解决方法：

1. 注册一个自定义的 ServletContextListener
2. 在 ServletContextListener 类, 初始化Hibernate会话，并将其存储到servlet上下文。
3. 在动作类, 可以通过servlet上下文的Hibernate会话，并执行任务正常的Hibernate操作。

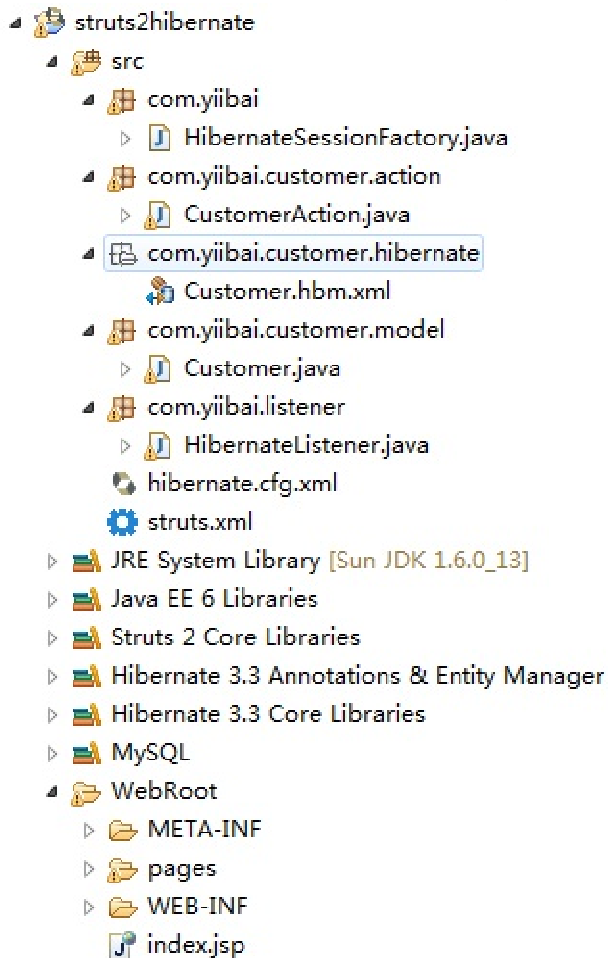
请参阅它们的关系：

```
Struts 2 <-- (Servlet Context) ---> Hibernate <-----> Database
```

在本教程中，在Struts中2开发我们显示了一个简单的客户模块(添加和列表功能), 并使用 Hibernate 进行数据库操作。使用上述部分机制集成(存储和检索在servlet上下文Hibernate的Session)。

## 1. 工程目录结构

来看看这个完整的项目文件夹结构。




## 2. MySQL表结构脚本

创建一个客户(customer)表。下面是SQL表脚本。

```
CREATE TABLE `customer` (
  `customer_id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `address` varchar(255) NOT NULL,
  `create_date` datetime NOT NULL,
  PRIMARY KEY (`customer_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

## 4. Hibernate 相关配置

Hibernate的模型和配置的东西。

Customer.java – 创建客户表对应的一个类。

```
package com.yiibai.customer.model;

import java.util.Date;

public class Customer implements java.io.Serializable {

    private Long customerId;
    private String name;
    private String address;
    private Date createdDate;

    //getter and setter methods
}
```

**Customer.hbm.xml** – Hibernate映射文件客户表。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.yiibai.customer.model.Customer"
    table="customer" catalog="yiibai">

        <id name="customerId" type="java.lang.Long">
            <column name="CUSTOMER_ID" />
            <generator class="identity" />
        </id>
        <property name="name" type="string">
            <column name="NAME" length="45" not-null="true" />
        </property>
        <property name="address" type="string">
            <column name="ADDRESS" not-null="true" />
        </property>
        <property name="createdDate" type="timestamp">
            <column name="CREATED_DATE" length="19" not-null="true'
        </property>
    </class>
</hibernate-mapping>
```

**hibernate.cfg.xml** – Hibernate数据库配置文件

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.bytecode.use_reflection_optimizer">fa
    <property name="hibernate.connection.password">password</proper
    <property name="hibernate.connection.url">jdbc:mysql://localhos
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLD
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="use_sql_comments">false</property>
    <mapping resource="com/yiibai/customer/hibernate/Customer.hbm.x
  </session-factory>
</hibernate-configuration>
```

# 5. Hibernate ServletContextListener

创建一个类 ServletContextListener, 并初始化Hibernate会话，并将其存储到servlet上下文。

**HibernateListener .java**

```
package com.yiibai.listener;

import java.net.URL;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateListener implements ServletContextListener{

    private Configuration config;
    private SessionFactory factory;
    private String path = "/hibernate.cfg.xml";
    private static Class clazz = HibernateListener.class;

    public static final String KEY_NAME = clazz.getName();

    public void contextDestroyed(ServletContextEvent event) {
      //
    }

    public void contextInitialized(ServletContextEvent event) {

     try {
            URL url = HibernateListener.class.getResource(path);
            config = new Configuration().configure(url);
            factory = config.buildSessionFactory();

            //save the Hibernate session factory into serlvet conte
            event.getServletContext().setAttribute(KEY_NAME, factor
      } catch (Exception e) {
             System.out.println(e.getMessage());
       }
    }
}
```

在 web.xml 文件中注册监听器。

**web.xml**

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Struts 2 Web Application</display-name>

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecu
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <listener>
    <listener-class>
      com.yiibai.listener.HibernateListener
    </listener-class>
  </listener>

</web-app>
```

# 6. Action

在动作类，可以通过servlet上下文的Hibernate会话和执行正常的Hibernate任务。

**CustomerAction.java**

```
package com.yiibai.customer.action;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.apache.struts2.ServletActionContext;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.yiibai.customer.model.Customer;
import com.yiibai.listener.HibernateListener;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
```

```
public class CustomerAction extends ActionSupport
    implements ModelDriven{

    Customer customer = new Customer();
    List<Customer> customerList = new ArrayList<Customer>();

    public String execute() throws Exception {
        return SUCCESS;
    }

    public Object getModel() {
        return customer;
    }

    public List<Customer> getCustomerList() {
        return customerList;
    }

    public void setCustomerList(List<Customer> customerList) {
        this.customerList = customerList;
    }

    //save customer
    public String addCustomer() throws Exception{

        //get hibernate session from the servlet context
        SessionFactory sessionFactory =
             (SessionFactory) ServletActionContext.getServletContex
                     .getAttribute(HibernateListener.KEY_NAME);

        Session session = sessionFactory.openSession();

        //save it
        customer.setCreatedDate(new Date());

        session.beginTransaction();
        session.save(customer);
        session.getTransaction().commit();

        //reload the customer list
        customerList = null;
        customerList = session.createQuery("from Customer").list();

        return SUCCESS;

    }

    //list all customers
    public String listCustomer() throws Exception{

        //get hibernate session from the servlet context
        SessionFactory sessionFactory =
             (SessionFactory) ServletActionContext.getServletContex
```

```
                     .getAttribute(HibernateListener.KEY_NAME);

        Session session = sessionFactory.openSession();

        customerList = session.createQuery("from Customer").list();

        return SUCCESS;

    }
}
```

## 7. JSP 页面

JSP页面用来添加和列出的客户。

**customer.jsp**

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>

<body>
<h1>Struts 2 + Hibernate integration example</h1>

<h2>Add Customer</h2>
<s:form action="addCustomerAction" >
  <s:textfield name="name" label="Name" value="" />
  <s:textarea name="address" label="Address" value="" cols="50" row
  <s:submit />
</s:form>

<h2>All Customers</h2>

<s:if test="customerList.size() > 0">
<table border="1px" cellpadding="8px">
    <tr>
        <th>Customer Id</th>
        <th>Name</th>
        <th>Address</th>
        <th>Created Date</th>
    </tr>
    <s:iterator value="customerList" status="userStatus">
        <tr>
            <td><s:property value="customerId" /></td>
            <td><s:property value="name" /></td>
            <td><s:property value="address" /></td>
            <td><s:date name="createdDate" format="dd/MM/yyyy" /></
        </tr>
    </s:iterator>
</table>
</s:if>
<br/>
<br/>

</body>
</html>
```

# 8. struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
  <constant name="struts.devMode" value="true" />

  <package name="default" namespace="/" extends="struts-default">

    <action name="addCustomerAction"
    class="com.yiibai.customer.action.CustomerAction" method="addCu
       <result name="success">pages/customer.jsp</result>
    </action>

    <action name="listCustomerAction"
    class="com.yiibai.customer.action.CustomerAction" method="listC
        <result name="success">pages/customer.jsp</result>
    </action>

  </package>
</struts>
```

# 9. 实例测试执行

访问客户模块：http://localhost:8080/struts2hibernate/listCustomerAction.action

localhost:8080/struts2hibernate/listCustomerAction.action

Struts2 + Hibernate 集成实例

添加顾客

Name: 易百教程

Address: 海南海口

Submit

所有顾客

| Customer Id | Name | Address | Created Date |
|---|---|---|---|
| 1 | yiibai | Hainan HaiKou | 02/12/2015 |

在名称和地址字段填写，点击提交按钮，插入的客户的详细信息会马上列出结果。

localhost:8080/struts2hibernate/addCustomerAction.action

Struts2 + Hibernate 集成实例

添加顾客

Name:

Address:

Submit

所有顾客

| Customer Id | Name | Address | Created Date |
|---|---|---|---|
| 1 | yiibai | Hainan HaiKou | 02/12/2015 |
| 2 | 易百教程 | 海南海口 | 02/12/2015 |

# 参考

1. Struts2 + Hibernate使用“Full Hibernate Plugin"集成
2. ServletContextListener 文档
3. Struts + Hibernate集成实例

代码下载 - http://pan.baidu.com/s/1hqhQJ7A

# Struts2+Spring+Hibernate集成实例 - Struts2教程

在本教程中，它显示的集成 “Struts2 + Spring + Hibernate“，请务必检查以下之前继续学习教程。

1. Struts2 + Hibernate集成实例
2. Struts2 + Spring 集成实例

参见集成步骤总结：

1. 获取所有的依赖库(很多)。
2. 注册 Spring 的 ContextLoaderListener 来整合 Struts2 和 Spring。
3. 使用 Spring 的 LocalSessionFactoryBean 来集成 Spring 和 Hibernate。
4. 完成所有连接。

请参阅它们之的关系：

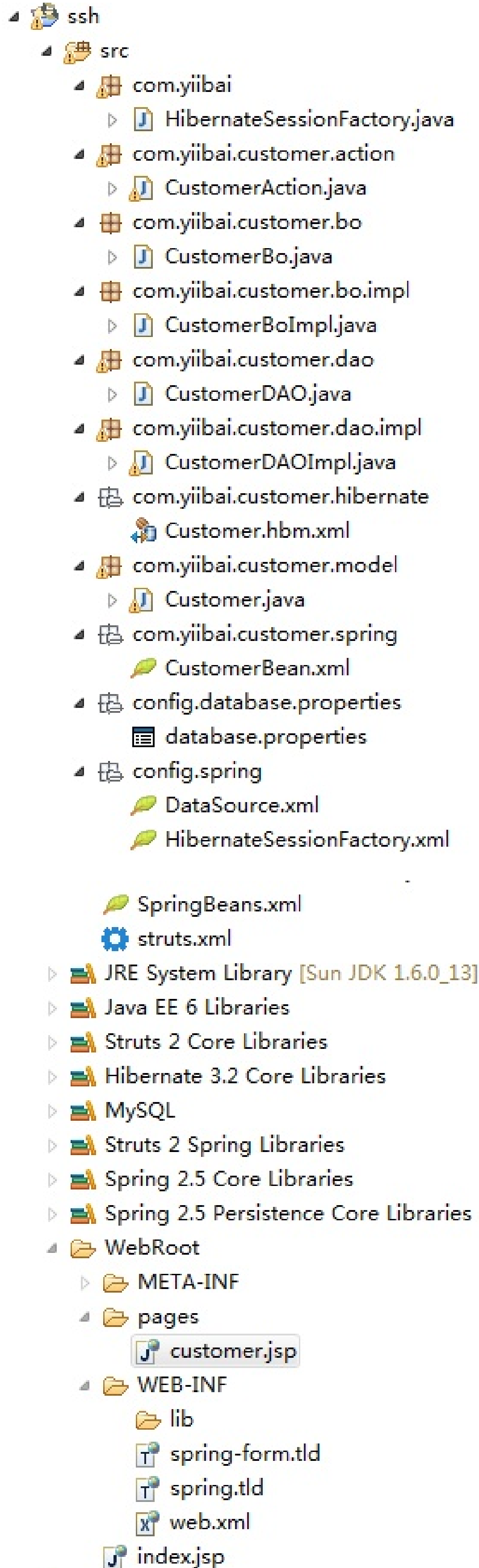
```
Struts 2 <-- (ContextLoaderListener) --> Spring <-- (LocalSessionFa
```

这将是一个很长的教程，相关解释并不是很多，请务必阅读上述2篇文章的详细情况说明以方面学习。

这将要创建一个客户页面，以添加客户和列表的自定义函数。前端使用Struts2显示，Spring作为依赖注入引擎，而 Hibernate 用来执行数据库操作。让我们开始...

## 1. 工程文件夹结构

在本章中，我们创建一个 ssh 的web工程，工程的目录结构如下图所示：




## 2. MySQL表结构结构

客户(customer)表脚本。

```
DROP TABLE IF EXISTS `yiibai`.`customer`;
CREATE TABLE  `yiibai`.`customer` (
  `CUSTOMER_ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `NAME` varchar(45) NOT NULL,
  `ADDRESS` varchar(255) NOT NULL,
  `CREATED_DATE` datetime NOT NULL,
  PRIMARY KEY (`CUSTOMER_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;
```

# 3. Hibernate相关配置

只有模型和映射文件是必需的，因为这里要用Spring处理Hibernate配置。

**Customer.java** – 创建客户表对应的一个类。

```
package com.yiibai.customer.model;

import java.util.Date;

public class Customer implements java.io.Serializable {

    private Long customerId;
    private String name;
    private String address;
    private Date createdDate;

    //getter and setter methods
}
```

**Customer.hbm.xml** – Hibernate的客户映射文件。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 20 Julai 2010 11:40:18 AM by Hibernate Tools 3.2.5.B
<hibernate-mapping>
    <class name="com.yiibai.customer.model.Customer"
        table="customer" catalog="yiibai">
        <id name="customerId" type="java.lang.Long">
            <column name="customer_id" />
            <generator class="identity" />
        </id>
        <property name="name" type="string">
            <column name="name" length="45" not-null="true" />
        </property>
        <property name="address" type="string">
            <column name="address" not-null="true" />
        </property>
        <property name="createdDate" type="timestamp">
            <column name="create_date" length="19" not-null="true"
        </property>
    </class>
</hibernate-mapping>
```

# 5. Struts2相关

实现了 Bo 和 DAO 设计模式。所有Bo和DAO将由Spring Spring bean配置文件注入。在DAO中，让它扩展Spring的HibernateDaoSupport来集成 Spring 和 Hibernate。

### CustomerBo.java

```
package com.yiibai.customer.bo;

import java.util.List;
import com.yiibai.customer.model.Customer;

public interface CustomerBo{

    void addCustomer(Customer customer);
    List<Customer> listCustomer();

}
```

### CustomerBoImpl.java

```
package com.yiibai.customer.bo.impl;

import java.util.List;
import com.yiibai.customer.bo.CustomerBo;
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.model.Customer;

public class CustomerBoImpl implements CustomerBo{

    CustomerDAO customerDAO;
    //DI via Spring
    public void setCustomerDAO(CustomerDAO customerDAO) {
        this.customerDAO = customerDAO;
    }

    //call DAO to save customer
    public void addCustomer(Customer customer){
        customerDAO.addCustomer(customer);
    }

    //call DAO to return customers
    public List<Customer> listCustomer(){
        return customerDAO.listCustomer();
    }
}
```

**CustomerDAO.java**

```
package com.yiibai.customer.dao;

import java.util.List;
import com.yiibai.customer.model.Customer;

public interface CustomerDAO{

    void addCustomer(Customer customer);
    List<Customer> listCustomer();

}
```

**CustomerDAOImpl.java**

```
package com.yiibai.customer.dao.impl;

import java.util.List;
import org.springframework.orm.hibernate3.support.HibernateDaoSuppo
import com.yiibai.customer.dao.CustomerDAO;
import com.yiibai.customer.model.Customer;

public class CustomerDAOImpl extends HibernateDaoSupport
    implements CustomerDAO{

    //add the customer
    public void addCustomer(Customer customer){
        getHibernateTemplate().save(customer);
    }

    //return all the customers in list
    public List<Customer> listCustomer(){
        return getHibernateTemplate().find("from Customer");
    }

}
```

**CustomerAction.java** – Struts2 的动作不再需要扩展ActionSupport，它将由 Spring 来处理。

```
package com.yiibai.customer.action;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import com.yiibai.customer.bo.CustomerBo;
import com.yiibai.customer.model.Customer;
import com.opensymphony.xwork2.ModelDriven;

public class CustomerAction implements ModelDriven{

    Customer customer = new Customer();
    List<Customer> customerList = new ArrayList<Customer>();

    CustomerBo customerBo;
    //DI via Spring
    public void setCustomerBo(CustomerBo customerBo) {
        this.customerBo = customerBo;
    }

    public Object getModel() {
        return customer;
    }
```

```
    public List<Customer> getCustomerList() {
        return customerList;
    }

    public void setCustomerList(List<Customer> customerList) {
        this.customerList = customerList;
    }

    //save customer
    public String addCustomer() throws Exception{

        //save it
        customer.setCreatedDate(new Date());
        customerBo.addCustomer(customer);

        //reload the customer list
        customerList = null;
        customerList = customerBo.listCustomer();

        return "success";

    }

    //list all customers
    public String listCustomer() throws Exception{

        customerList = customerBo.listCustomer();

        return "success";

    }

}
```

# 6. Spring相关配置

几乎所有的配置都是在这里完成是由Spring专门来整合。

**CustomerBean.xml** – 声明 Spring 的 bean：Action, BO 和 DAO.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xs

     <bean id="customerAction" class="com.yiibai.customer.action.Cu
        <property name="customerBo" ref="customerBo" />
    </bean>

    <bean id="customerBo" class="com.yiibai.customer.bo.impl.Custon
        <property name="customerDAO" ref="customerDAO" />
    </bean>

       <bean id="customerDAO" class="com.yiibai.customer.dao.impl.C
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

</beans>
```

**database.properties** – 声明数据库详细信息

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/yiibai
jdbc.username=root
jdbc.password=password
```

**DataSource.xml** – 创建一个数据库源的Bean

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

 <bean
   class="org.springframework.beans.factory.config.PropertyPlaceho
   <property name="location">
     <value>WEB-INF/classes/config/database/properties/database.pro
   </property>
</bean>

  <bean id="dataSource"
         class="org.springframework.jdbc.datasource.DriverManagerDa
    <property name="driverClassName" value="${jdbc.driverClassName
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
  </bean>

</beans>
```

**HibernateSessionFactory.xml** – 创建一个SessionFactory Bean来集成Spring和Hibernate。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<!-- Hibernate session factory -->
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBe

    <property name="dataSource">
      <ref bean="dataSource"/>
    </property>

    <property name="hibernateProperties">
       <props>
         <prop key="hibernate.dialect">org.hibernate.dialect.MySQLD
         <prop key="hibernate.show_sql">true</prop>
       </props>
    </property>

    <property name="mappingResources">
        <list>
          <value>com/yiibai/customer/hibernate/Customer.hbm.xml</va
        </list>
    </property>

</bean>
</beans>
```

**SpringBeans.xml** – 创建一个核心 Spring 的 bean 配置文件，作为中央的 bean 管理层。

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <!-- Database Configuration -->
    <import resource="config/spring/DataSource.xml"/>
    <import resource="config/spring/HibernateSessionFactory.xml"/>

    <!-- Beans Declaration -->
    <import resource="com/yiibai/customer/spring/CustomerBean.xml"/

</beans>
```

# 7. JSP 页面

JSP页面来显示使用 Struts2 标签的元素。

### customer.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>

<body>
<h1>Struts 2 + Spring + Hibernate integration example</h1>

<h2>Add Customer</h2>
<s:form action="addCustomerAction" >
  <s:textfield name="name" label="Name" value="" />
  <s:textarea name="address" label="Address" value="" cols="50" row
  <s:submit />
</s:form>

<h2>All Customers</h2>

<s:if test="customerList.size() > 0">
<table border="1px" cellpadding="8px">
    <tr>
        <th>Customer Id</th>
        <th>Name</th>
        <th>Address</th>
        <th>Created Date</th>
    </tr>
    <s:iterator value="customerList" status="userStatus">
        <tr>
            <td><s:property value="customerId" /></td>
            <td><s:property value="name" /></td>
            <td><s:property value="address" /></td>
            <td><s:date name="createdDate" format="dd/MM/yyyy" /></
        </tr>
    </s:iterator>
</table>
</s:if>
<br/>
<br/>

</body>
</html>
```

# 8. struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
     <constant name="struts.devMode" value="true" />

    <package name="default" namespace="/" extends="struts-default">

        <action name="addCustomerAction"
            class="customerAction" method="addCustomer" >
            <result name="success">pages/customer.jsp</result>
        </action>

        <action name="listCustomerAction"
            class="customerAction" method="listCustomer" >
            <result name="success">pages/customer.jsp</result>
        </action>

    </package>

</struts>
```

# 9. Struts 2 + Spring

要集成Struts2和Spring，只需注册ContextLoaderListener监听器类，定义一个“contextConfigLocation”参数要求Spring容器来解析“SpringBeans.xml”，而不使用默认的“applicationContext.xml”。

**web.xml**

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Struts 2 Web Application</display-name>

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecu
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/classes/SpringBeans.xml</param-value>
  </context-param>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

</web-app>
```

## 10. 运行实例

在浏览器中打开网址：http://localhost:8080/ssh/listCustomerAction.action

## 参考

1. Struts2 + Hibernate集成实例

2. Struts2 + Spring集成实例
3. Struts2 + Hibernate 使用 Full Hibernate Plugin插件集成

代码下载 - http://pan.baidu.com/s/1mgzt1Xm (含ssh相关类库，详见 lib 目录，文件大小约：18M)。

# Struts2+Log4j集成 - Struts2教程

在本教程中，我们学习如何将log4j框架在Struts2的Web应用程序集成。所有需要做的有：

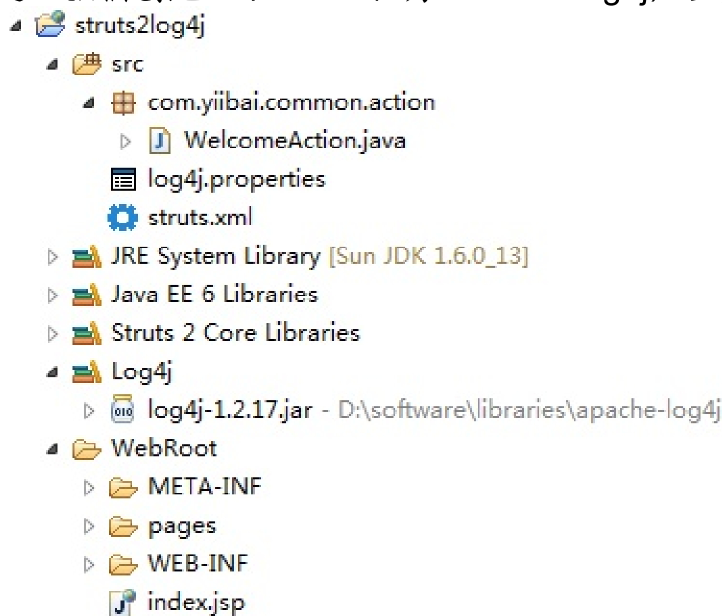
1. 包含 log4j.jar 作为项目依赖
2. 创建一个 log4j.properties 文件，并把它放入 classpath 的根目录-放到资源文件夹中。

相关技术和工具的使用：

1. Log4j 1.2.17
2. Struts 2.1.8
3. Tomcat 6
4. MyEclipse 10

## 1. 工程结构

这里我们创建一个web工程为：struts2log4j，参见下面最终的工程结构：




## 2. log4j.properties

创建log4j的属性文件，并把它放入资源文件夹，请参阅步骤#1。log4j.properties

```
# Root logger option
log4j.rootLogger=ERROR, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm

# Redirect log messages to a log file, support rolling backup file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=${catalina.home}/logs/mystruts2app.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss
```

## 4. Struts2 Action 和 Logging

一个简单的动作返回一个页面，并显示了如何来执行 log4j 消息日志记录。
WelcomeAction.java

```
package com.yiibai.common.action;

import org.apache.log4j.Logger;
import com.opensymphony.xwork2.ActionSupport;

public class WelcomeAction extends ActionSupport {

    private static final long serialVersionUID = 1L;

    //get log4j
    private static final Logger logger = Logger.getLogger(WelcomeAc

    public String execute() throws Exception {

        // logs debug message
        if (logger.isDebugEnabled()) {
            logger.debug("execute()!");
        }

        // logs exception
        logger.error("This is Error message", new Exception("Testir

        return SUCCESS;

    }
}
```

# 5. Struts2配置

Struts2 的配置和JSP页面，如果想了解的话。struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <constant name="struts.devMode" value="true" />

    <package name="welcome" namespace="/" extends="struts-default">

        <action name="welcome" class="com.yiibai.common.action.Welc
            <result name="success">pages/success.jsp</result>
        </action>

    </package>

</struts>
```

web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">

    <display-name>Struts 2 Web Application</display-name>

    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAr
        </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

pages/success.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
</head>

<body>
<h1>Struts 2 + Log4j integration example</h1>

</body>
</html>
```

# 6. 实例测试

运行Struts 2的Web应用程序，并访问welcome的动作。

在浏览器中打开 URL : http://localhost:8888/struts2log4j/welcome

*6.1* 所有日志消息将显示在控制台中。

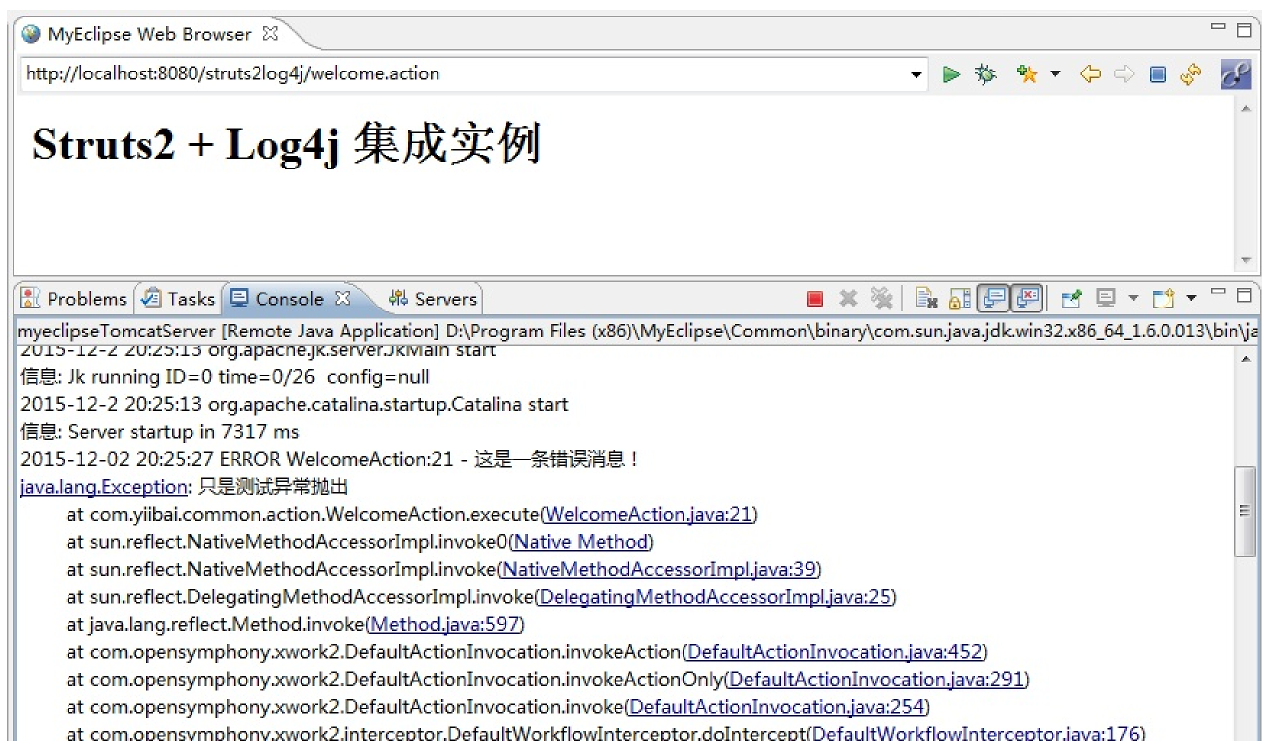



*Figure : Eclipse* 终端

*6.2* 此外，日志文件将在Tomcat 的日志文件夹中被创建。

图片: *C:\mystruts2app.log*

下载代码 – http://pan.baidu.com/s/1nt7yVep

# Struts2配置Action类的静态参数 - Struts2教程

在某些情况下，可能需要一个Action类分配一些预定义或静态的参数值。

## 为动作定义静态参数

在Struts2，可以在 struts.xml 文件中的通过标记进行配置，例如，

**struts.xml**

```
<struts>

   <constant name="struts.custom.i18n.resources" value="global" />
   <constant name="struts.devMode" value="true" />

   <package name="default" namespace="/" extends="struts-default">
    <action name="locale" class="com.yiibai.common.action.LocaleAct
        <result name="SUCCESS">pages/welcome.jsp</result>
        <param name="EnglishParam">English</param>
                <param name="ChineseParam">Chinese</param>
                 <param name="FranceParam">France</param>
    </action>
   </package>
</struts>
```

它分配三个预定义的参数值到LocaleAction Action类。

## 从动作获取静态参数

要从struts.xml中获取静态参数值，Action类必须实现参数化Parameterizable接口。动作的静态参数是由staticParams拦截，其中包括在默认堆栈控制

动作的静态参数是由staticParams拦截，包括在默认堆栈 “struts-default.xml” 中控制。

## 1. Map属性

在操作类初始化期间，staticParams拦截器将通过动作类的setParams()方法获取预先定义的参数值。

```
//...
import com.opensymphony.xwork2.config.entities.Parameterizable;

public class LocaleAction implements Parameterizable{

    Map<String, String> params;
    //...
    public void setParams(Map<String, String> params) {
        this.params = params;
    }
}
```

## 2. JavaBean 属性

在动作类的初始化，如果创建了getter和setter方法得当，staticParams拦截器将设置预先定义的参数值，以对应于该“参数”的每JavaBean属性。

```
//...
import com.opensymphony.xwork2.config.entities.Parameterizable;

public class LocaleAction implements Parameterizable{

    String englishParam;
    String chineseParam;
    String franceParam;

    public String getEnglishParam() {
        return englishParam;
    }

    public void setEnglishParam(String englishParam) {
        this.englishParam = englishParam;
    }

    public String getChineseParam() {
        return chineseParam;
    }

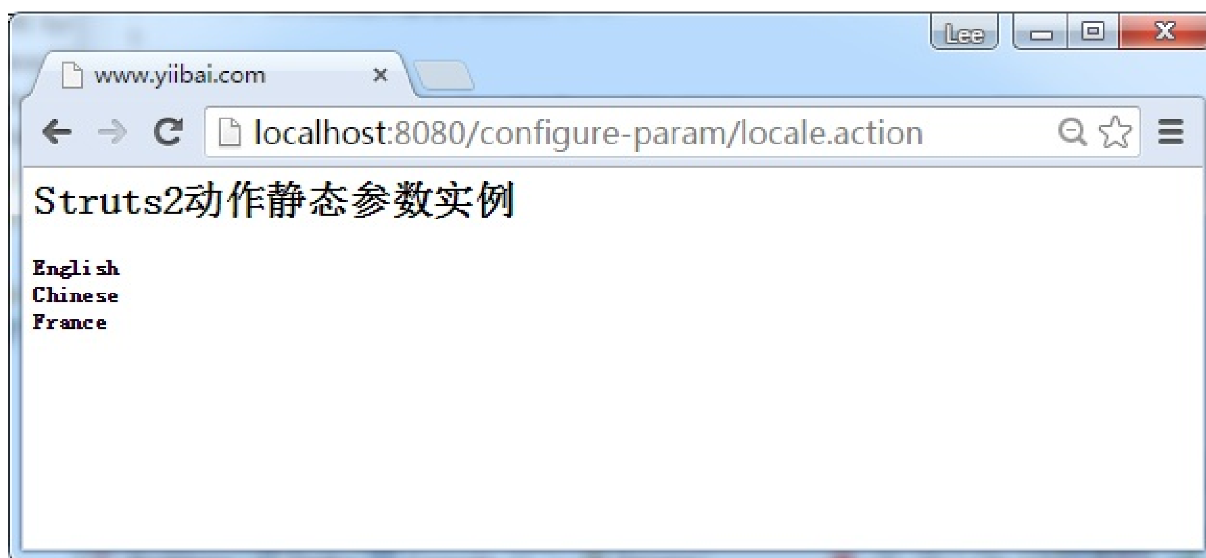
    public void setChineseParam(String chineseParam) {
        this.chineseParam = chineseParam;
    }

    public String getFranceParam() {
        return franceParam;
    }

    public void setFranceParam(String franceParam) {
        this.franceParam = franceParam;
    }
    //...
}
```

## 2. 运行实例

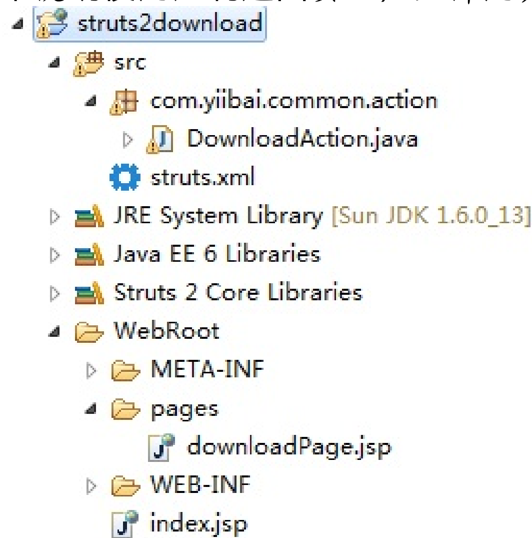
在浏览器中打开URL：http://localhost:8080/configure-param/locale.action




代码下载(configure-param) - http://pan.baidu.com/s/1dDmGDK9

# Struts2下载文件实例 - Struts2教程

这是一个Struts2的例子来说明使用定制返回类型，允许用户下载文件。web工程的




文件夹结构如下所示：

## 1. Action

在Action类中，声明一个 InputStream 的数据类型和getter方法。

**DownloadAction.java**

```
package com.yiibai.common.action;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import com.opensymphony.xwork2.ActionSupport;

public class DownloadAction extends ActionSupport{

    private InputStream fileInputStream;

    public InputStream getFileInputStream() {
        return fileInputStream;
    }

    public String execute() throws Exception {
        fileInputStream = new FileInputStream(new File("C:\\file-fo
    }
}
```

# 2. 视图文件

一个正常的页面，有一个下载链接，用于下载文件。

**downloadPage.jsp**

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>

<body>
<h1>Struts 2 download file example</h1>

<s:url id="fileDownload" namespace="/" action="download" ></s:url>

<div><div class="ads-in-post hide_if_width_less_800">
<script async src="//pagead2.googlesyndication.com/pagead/js/adsbyg
<!-- 728x90 - After2ndH4 -->
<ins class="adsbygoogle hide_if_width_less_800"
     style="display:inline-block;width:728px;height:90px"
     data-ad-client="ca-pub-2836379775501347"
     data-ad-slot="3642936086"
     data-ad-region="yiibairegion"></ins>
<script>
(adsbygoogle = window.adsbygoogle || []).push({});
</script>
</div></div><h2>Download file - <s:a href="%{fileDownload}">fileAB(
</h2>

</body>
</html>
```

# 3. struts.xml

定义下载文件的细节。 <param name="inputName"> 值是从Action的InputStream属性的名称。

阅读Struts2的数据流结果文档以了解更详细信息。

**struts.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<constant name="struts.devMode" value="true" />

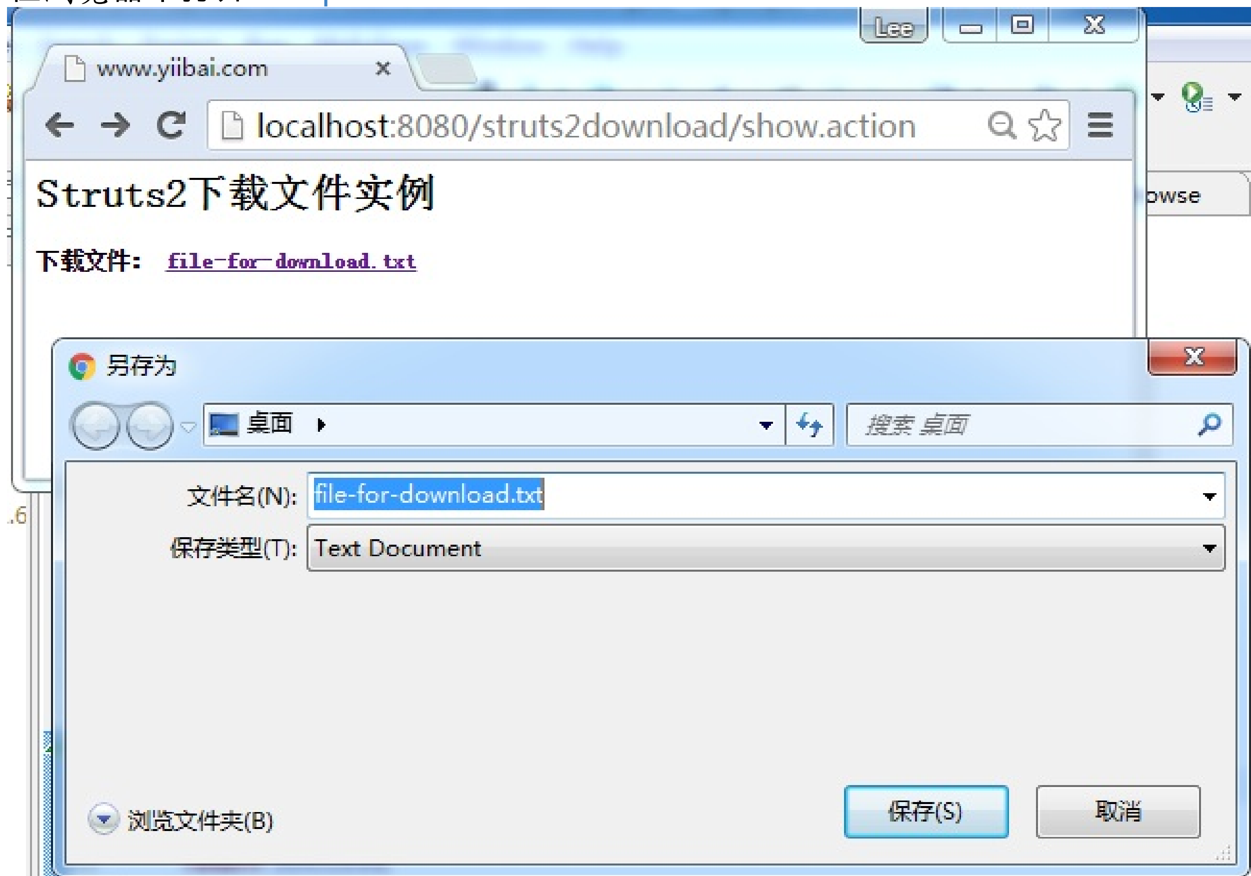
<package name="default" namespace="/" extends="struts-default">
   <action name="show">
    <result name="success">pages/downloadPage.jsp</result>
   </action>

   <action name="download" class="com.yiibai.common.action.Downloac
    <result name="success" type="stream">
      <param name="contentType">application/octet-stream</param>
      <param name="inputName">fileInputStream</param>
      <param name="contentDisposition">attachment;filename="file-fc
    </result>
   </action>
</package>

</struts>
```

## 4. 执行结果

在浏览器中打开：http://localhost:8080/struts2download/




# 参考

1. http://struts.apache.org/2.x/docs/stream-result.html
2. http://www.iana.org/assignments/media-types/
3. http://www.yiibai.com/struts/struts-download-file-from-website-example.html
4. http://www.yiibai.com/java/how-to-download-file-from-website-java-jsp.html
5. http://struts.apache.org/2.x/docs/how-can-we-return-a-text-string-as-the-response.html

代码下载(struts2download) - http://pan.baidu.com/s/1jGg0Lzo

# Struts2和JSON实例 - Struts2教程

在这个Struts2例子，将学习如何通过“struts2-json-plugin.jar”库将对象转换为JSON格式的数据。

## 1. Action (JSON)

这是一个将被转换成JSON格式的 Action 类。

```
package com.yiibai.common.action;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.opensymphony.xwork2.Action;

public class JSONDataAction{

    private String string1 = "A";
    private String[] stringarray1 = {"A1","B1"};
    private int number1 = 123456789;
    private int[] numberarray1 = {1,2,3,4,5,6,7,8,9};
    private List<String> lists = new ArrayList<String>();
    private Map<String, String> maps = new HashMap<String, String>(

    //no getter method, will not include in the JSON
    private String string2 = "B";

    public JSONDataAction(){
        lists.add("list1");
        lists.add("list2");
        lists.add("list3");
        lists.add("list4");
        lists.add("list5");

        maps.put("key1", "value1");
        maps.put("key2", "value2");
        maps.put("key3", "value3");
        maps.put("key4", "value4");
        maps.put("key5", "value5");
    }

    public String execute() {
                return Action.SUCCESS;
        }
```

```
    public String getString1() {
        return string1;
    }

    public void setString1(String string1) {
        this.string1 = string1;
    }

    public String[] getStringarray1() {
        return stringarray1;
    }

    public void setStringarray1(String[] stringarray1) {
        this.stringarray1 = stringarray1;
    }

    public int getNumber1() {
        return number1;
    }

    public void setNumber1(int number1) {
        this.number1 = number1;
    }

    public int[] getNumberarray1() {
        return numberarray1;
    }

    public void setNumberarray1(int[] numberarray1) {
        this.numberarray1 = numberarray1;
    }

    public List<String> getLists() {
        return lists;
    }

    public void setLists(List<String> lists) {
        this.lists = lists;
    }

    public Map<String, String> getMaps() {
        return maps;
    }

    public void setMaps(Map<String, String> maps) {
        this.maps = maps;
    }

}
```

# 3. struts.xml

要输出JSON数据，需要声明一个包，它扩展“json-default”，会将结果类型转为“json”。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

   <constant name="struts.devMode" value="true" />

   <package name="default" namespace="/" extends="json-default">
      <action name="getJSONResult"
           class="com.yiibai.common.action.JSONDataAction">
              <result type="json" />
      </action>
    </package>

</struts>
```

# 4. 实例

访问动作URL时，JSONDataAction属性将被转换成JSON格式。

*http://localhost:8080/struts2json/getJSONResult.action*




JSON 格式 ...

```
{
   "lists":["list-1","list-2","list-3","list-4","list-5"],
   "maps":
   {
     "key4":"value4","key3":"value3","key5":"value5","key2":"value2
   },
   "number1":123456789,
   "numberarray1":[1,2,3,4,5,6,7,8,9],
   "string1":"A",
   "stringarray1":["A1","B1"]
}
```

希望这个简单的例子可以了解JSON插件在Struts2是如何工作的有一个总体的思路。不过，还是有很多有用的参数不包括在这里，请务必阅读 Struts2 JSON插件文档 以获取更多详细信息。下载源代码(struts2json) – http://pan.baidu.com/s/1bnv8l9X

# SWING 教程

JAVA 提供了一组丰富的平台无关的方式来创建图形用户界面的库。在这篇文章中，我们将学习Swing GUI控件。

# 读者

本教程是专为愿意学习JAVA GUI编程软件专业人员提供简单轻松的学习步骤。本教程会给你很大的JAVA GUI编程概念的理解，并完成本教程后可以把自己的专业知识技术提升到一个较高的水平。

# 前提条件

继续本教程之前，你应该有一个基本的了解Java编程语言，文本编辑器和执行程序等

# Swing介绍 - Swing

Swing API 可扩展 GUI组件，以减轻开发者的生活创造基于JAVA前端/GUI应用。它是建立在AWT API之上，并作为 AWT API 的更换，因为它几乎每一个控制对应AWT控制。 Swing 组件遵循模型 - 视图 - 控制器架构，以满足以下标准。

- 一个单一的 API 是足够支持多种外观和风格。
- API 模拟驱动，使最高级别的API不要求有数据。
- API 使用Java Bean的模式，使生成工具和IDE可以提供更好的服务给开发者使用它。

## MVC架构

Swing API架构如下松散的，基于MVC架构，以下列方式支付。

- 模型表示组件的数据。
- 查看代表组件的数据可视化表示形式。
- 控制器采用的视图上的用户输入，并在组件的数据的变化反映。
- Swing 组件模型作为一个单独的元素和景观，是棒状控制器部分用户界面中的元素。使用这种方式，Swing具有可插拔的外观与风格架构。

## Swing 特点

- 重量轻 - Swing 组件是独立的原生操作系统的API与Swing API控件呈现大多采用纯JAVA代码，而不是底层的操作系统调用。
- 丰富的控件 - Swing 提供了一套丰富的先进的控制系统，如树，JTabbedPane，滑块，颜色选择器，表格控件
- 高度可定制 - Swing 控件可以定制视觉外观是非常简单的方法，独立的内部表示。
- 可插拔的外观和感觉 - 基于Swing GUI应用程序外观和风格基于可用值，可以在运行时改变。

# Swing开发环境安装 - Swing

本节将指导如何下载和设置Java 在您的机器上。请按照下列步骤来设置环境。

Java SE 是免费的，提供的链接下载Java。所以根据您的操作系统，下载一个版本。

按照说明下载 java 和运行 .exe 在你的机器上安装Java。一旦在机器上安装了Java，还需要设置环境变量指向正确的安装目录：

## 对于Windows 2000/XP/win7 路径：

假设您已经安装了Java在c:Program Filesjavajdk 目录：

- 右键单击 '我的电脑r' 并选择 '属性'。
- 点击 '环境变量' 按钮下 '高级' 标签。
- 现在在 '路径' 变量，它也包含Java可执行文件的路径。 例如，如果路径当前设置为 'C:WINDOWSSYSTEM32', 然后可更改你的路径为 'C:WINDOWSSYSTEM32;c:Program Filesjavajdkin'.

## 适用于Linux，UNIX，Solaris和FreeBSD的路径：

应设置环境变量PATH指向Java二进制文件已经安装。如果你这样做有困难，请参阅shell文件。

例如，如果使用bash作为shell，添加以下行到结束行 “.bashrc文件中： '.bashrc: export PATH=/path/to/java:$PATH'

## 流行的Java编辑器：

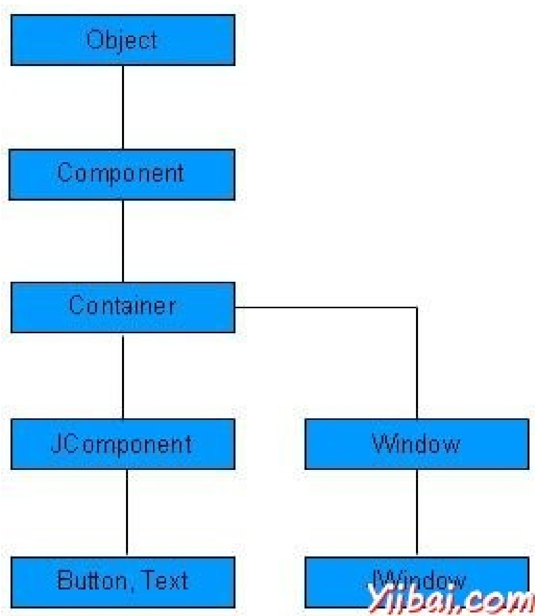
编写Java程序，需要一个好的文本编辑器。在市场上有更复杂的 IDE。但现在，可以考虑以下两种：

- 记事本: 在Windows机器上，你可以使用任何简单的文本编辑器（如记事本）（本教程推荐TextPad）。
- Netbeans：是一个Java IDE，它是开源和免费，可下载 http://www.netbeans.org/index.htmll.
- Eclipse : 也是一个Java IDE Eclipse开源社区开发的，可以直接从网上下载 http://www.eclipse.org/.

# Swing控件 - Swing

每一个用户界面参考主要有以下三个方面：

- UI 元素: 有核心视觉元素，最终用户看到并与之交互。 GWT 提供了一个巨大的名单变化更复杂，本教程我们从基本的广泛使用和常见的元素。
- 布局: 他们定义 UI 元素如何应安排在屏幕上，并提供一个最终的外观和感觉的GUI（图形用户界面）。在“布局”一章，这部分将被覆盖。
- 行为: 这些事件发生时，与用户交互的 UI 元素。这部分将被覆盖在事件处理“一章。




每个Swing 控件继承属性从以下组件的类层次结构。

| Sr. No. | 类 & 描述 |
| --- | --- |
| 1 | Component 容器是一个抽象基类的非菜单用户界面控件摆动。部分指图形表示的对象 |
| 2 | Container Container是一个组件，它可以包含其他Swing组件。 |
| 3 | JComponent JComponent是一个基类，所有Swing UI 组件。为了使用继承自JComponent 的 swing组件，组件必须是一个包容层次结构，其根是一个顶层的Swing容器。 |

## SWING UI 元素:

以下是常用的控件列表而设计的图形用户界面使用Swing。

| Sr. No. | 控件& 描述 |
|---|---|
| 1 | JLabel 一个JLabel对象是在容器中放置文本的一个组成部分。 |
| 2 | JButton 该类创建标记的按钮。 |
| 3 | JColorChooser JColorChooser提供一个用于控制窗格设计，让用户操作和选择颜色。 |
| 4 | JCheck Box JCheckBox的是一个图形化的组件可以在一个（true）或关闭（false）状态时。 |
| 5 | JRadioButton JRadioButton类是一个图形化的组件可以在一个（true）或关闭（false）状态时。在一组。 |
| 6 | JList JList组件向用户展示一个滚动的文本项列表。 |
| 7 | JComboBox JComboBox组件为用户提供了一个选择显示菜单。 |
| 8 | JTextField JTextField的对象是一个文本组件，它允许编辑的单行文本。 |
| 9 | JPasswordField JPasswordField中对象是一个专门用于输入密码的文本组件。 |
| 10 | JTextArea JTextArea对象是一个文本组件，它允许编辑的多行文本。 |
| 11 | ImageIcon ImageIcon的控件的图标界面，实现从图像绘制图标 |
| 12 | JScrollbar Scrollbar控件代表一个滚动条组件以让用户从范围选择的值。 |
| 13 | JOptionPane JOptionPane的规定设置标准对话框，提示用户提供值或向其发出通知。 |
| 14 | JFileChooser JFileChooser的控制代表一个对话框窗口，用户可以选择一个文件。 |
| 15 | JProgressBar 随着任务的进展，进度条接近完成显示任务的完成百分比。 |
| 16 | JSlider JSlider让用户以图形界的时间间隔内滑动旋钮选择一个值。 |
| 17 | JSpinner JSpinner让用户从一个有序序列中选择一个数字或者一个对象值的单行输入字段。 |

# Swing事件处理 - Swing

## 什么是事件？

一个对象的状态变化被称为事件，即事件描述源状态的变化。事件产生的结果与用户交互的图形用户界面组件。例如，点击一个按钮，移动鼠标，通过键盘输入一个字符，从列表中选择一个项目，滚动页面的活动，使一个事件的发生。

## 事件的类型

事件可以被大致分为两类：

- 前台事件 - 这些事件需要用户直接互动。在图形用户界面中的图形组件交互的人产生的后果。例如，点击一个按钮，移动鼠标，通过键盘输入一个字符，从列表中选择一个项目，滚动页面等
- 后台事件 - 这些事件，需要最终用户的交互是已知的作为背景的事件。操作系统的中断，硬件或软件故障，定时器到期时，操作完成的背景事件的例子。

## 事件处理是什么？

事件处理机制，控制的事件，并决定如果一个事件发生时，会发生什么。这种机制被称为事件处理程序，在事件发生时执行的代码。 Java使用代理事件模型来处理事件。该模型定义了标准的机制来生成和处理事件。让我们简要介绍这种模式。

代理事件模型具有以下的主要参与者，即：

- 源 - 源是一个对象，在该对象上的事件发生。它的处理器提供发生事件的信息来源是可靠的。 JAVA提供与源对象的类。
- 监听器 - 它也被称为作为事件handler.Listener是负责产生响应一个事件。从Java实现的角度来看，监听器也是一个对象。等待直到它接收到一个事件监听器。一旦收到事件，监听器进程的事件就返回。

这种方法的好处是，用户界面逻辑完全分开，生成该事件的逻辑。用户界面元素是能够委派的事件处理单独的一段代码。在这个模型中，需要与源对象注册监听使侦听器能够接收事件通知。这是一个有效的方式处理事件，因为这些事件通知只发送给那些监听器要接收他们。

## 参与事件处理的步骤

- 用户单击该按钮时产生该事件。

- 现在有关事件类的对象是自动创建的信息源和事件在同一对象得到填充。
- 事件对象被转发注册监听器类的方法。
- 该方法现在得到执行并返回。

# 要记住的要点有关监听器

- 为了设计一个监听类，我们必须开发一些监听器接口。这些监听器接口预测一些公共的抽象监听器类必须实现的回调方法。
- 如果不执行任何预定义的接口，类不能作为源对象的监听器类。

# 回调方法

这些方法所提供的API提供程序，被定义为应用程序员和应用程序开发者调用。这里的回调方法代表一个事件的方法。在响应一个事件的Java JRE将触发回调方法。所有这些回调方法的监听器接口。

如果一个组件需要一些监听器会听的事件源必须注册自己监听。

# 事件处理例子

选择使用任何编辑器创建以下java程序在 D:/ > SWING > com > yiibai > gui >

SwingControlDemo.java

```
package com.yiibai.gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
      swingLayoutDemo.showEventDemo();
   }
```

```
   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showEventDemo(){
      headerLabel.setText("Control in action: Button");

      JButton okButton = new JButton("OK");
      JButton submitButton = new JButton("Submit");
      JButton cancelButton = new JButton("Cancel");

      okButton.setActionCommand("OK");
      submitButton.setActionCommand("Submit");
      cancelButton.setActionCommand("Cancel");

      okButton.addActionListener(new ButtonClickListener());
      submitButton.addActionListener(new ButtonClickListener());
      cancelButton.addActionListener(new ButtonClickListener());

      controlPanel.add(okButton);
      controlPanel.add(submitButton);
      controlPanel.add(cancelButton);

      mainFrame.setVisible(true);
   }

   private class ButtonClickListener implements ActionListener{
      public void actionPerformed(ActionEvent e) {
         String command = e.getActionCommand();
         if( command.equals( "OK" ))  {
            statusLabel.setText("Ok Button clicked.");
         }
         else if( command.equals( "Submit" ) )  {
```

```
                statusLabel.setText("Submit Button clicked.");
            }
            else  {
                statusLabel.setText("Cancel Button clicked.");
            }
        }
    }
}
```

编译程序，使用命令提示符。到 D:/ > SWING 然后输出以下命令。

```
D:AWT>javac comyiibaiguiSwingControlDemo.java
```

如果没有错误出现，这意味着编译成功。使用下面的命令来运行程序。

```
D:AWT>java com.yiibai.gui.SwingControlDemo
```

验证下面的输出



# Swing事件监听器 - Swing

事件监听器代表负责处理事件的接口。 Java 提供了各种事件监听器类，但我们将讨论那些被更频繁地使用。每一个事件监听器方法具有方法的EventObject类的子类的对象，这是作为一个单独的参数。例如，鼠标事件侦听器方法将接受MouseEvent的实例，其中派生的事件的EventObject。

## EventListner 接口

它是一个标记接口，每一个监听器接口扩展。这个类定义在java.util包。

## 类声明

以下是声明java.util.EventListener接口：

```
public interface EventListener
```

## Swing事件监听器接口：

以下是常用的的事件监听器列表。

| Sr. No. | Control & Description |
|---|---|
| 1 | ActionListener 该接口用于接收动作事件。 |
| 2 | ComponentListener 该接口用于接收组件事件。 |
| 3 | ItemListener 该接口用于接收项目事件。 |
| 4 | KeyListener 该接口用于接收按键事件。 |
| 5 | MouseListener 该接口用于接收鼠标事件。 |
| 6 | WindowListener 该接口用于接收窗口事件。 |
| 7 | AdjustmentListener 该接口用于接收调整事件。 |
| 8 | ContainerListener 该接口用于接收容器事件。 |
| 9 | MouseMotionListener 此接口用于接收鼠标移动事件。 |
| 10 | FocusListener 该接口用于接收焦点事件。 |

# Swing事件适配器 - Swing

适配器是抽象类，用于接收各种事件。这些类中的方法是空的。这些类存在的目的是方便创建监听器对象。

## SWING适配器：

以下是常用的适配器列表，监听SWING GUI事件。

| Sr. No. | 适配器&说明 |
|---|---|
| 1 | FocusAdapter 接收焦点事件的抽象适配器类。 |
| 2 | KeyAdapter 接收按键事件的抽象适配器类。 |
| 3 | MouseAdapter 接收鼠标事件的抽象适配器类。 |
| 4 | MouseMotionAdapter 接收鼠标移动事件的抽象适配器类。 |
| 5 | WindowAdapter 接收窗口事件的抽象适配器类。 |

# Swing Layout布局 - Swing

布局的意味着，在容器内的配置的组件。在其他的方式，我们可以认为在一个特定的容器内的位置放置组件。布局管理器所控制布点的任务是自动完成的。

## 布局管理器

布局管理器自动定位容器内的所有组件。如果我们不使用布局管理器，然后定位组件的默认布局管理器。这是可能的手工布局的控制，但由于以下两个原因，它变得非常困难。

- 这是非常繁琐的容器内处理大量的控制。
- 通常当我们需要安排他们没有给出一个组件的宽度和高度信息。

Java 为我们提供了各种布局管理器来定位控制。属性如大小，形状和排列变化从一个布局管理器，其他的布局管理器。的小应用程序或应用程序窗口的大小改变时，即布局管理器applet浏览器或应用程序窗口的尺寸适应于响应的大小，形状和排列的组件也随之变化。

布局管理器关联的与每个容器对象。每一个布局管理器是实现布局管理接口的类的一个对象。

以下是接口定义布局管理器的功能。

| Sr. No. | 接口&说明 |
|---|---|
| 1 | LayoutManager 布局管理器接口声明的对象将充当一个布局管理器类需要实现的方法。 |
| 2 | LayoutManager2 LayoutManager2中的子接口布局管理。这个接口是为那些知道如何布局容器的基础上布局约束对象的类。 |

## AWT布局管理器类 ：

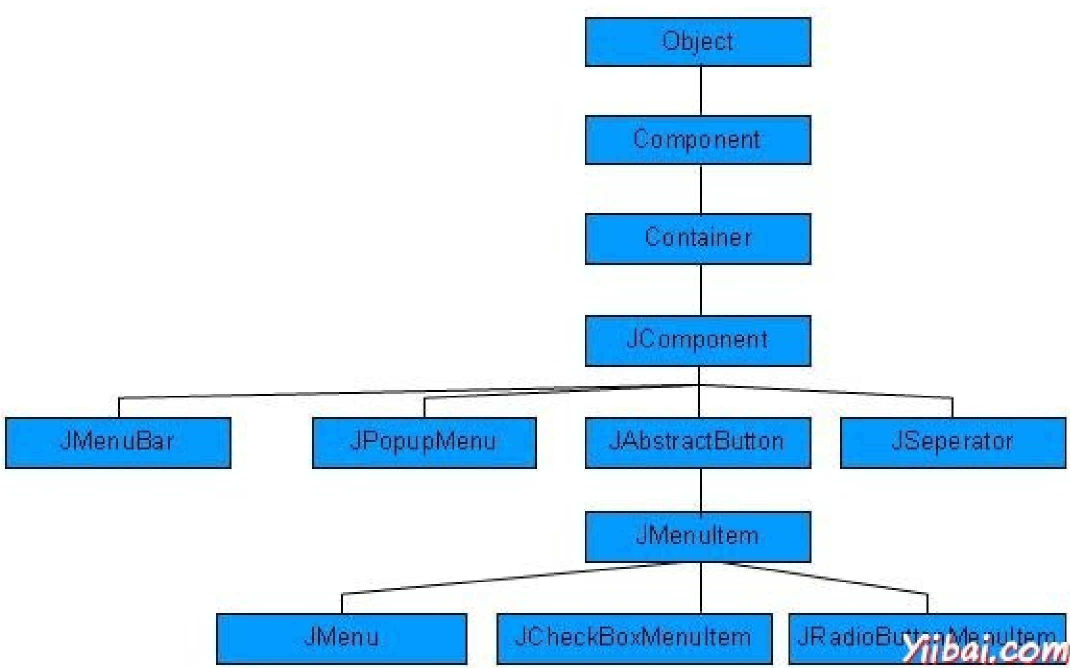
以下是常用的控件列表而设计的图形用户界面使用AWT。

| Sr. No. | LayoutManager & Description |
|---|---|
| 1 | BorderLayout The borderlayout arranges the components to fit in the five regions: east, west, north, south and center. |
| 2 | CardLayout The CardLayout object treats each component in the container as a card. Only one card is visible at a time. |
| 3 | FlowLayout The FlowLayout is the default layout.It layouts the components in a directional flow. |
| 4 | GridLayout The GridLayout manages the components in form of a rectangular grid. |
| 5 | GridBagLayout This is the most flexible layout manager class.The object of GridBagLayout aligns the component vertically,horizontally or along their baseline without requiring the components of same size. |
| 6 | GroupLayout The GroupLayout hierarchically groups components in order to position them in a Container. |
| 7 | SpringLayout A SpringLayout positions the children of its associated container according to a set of constraints. |

# Swing Menu菜单类 - Swing

正如我们所知道与菜单相关的每一个顶层窗口有一个菜单栏。此菜单栏包括提供给最终用户的各种菜单的选择。此外，每个选择包含这就是所谓的下拉菜单中的选项列表。菜单和菜单项的控件是MenuComponent类的子类。

## 菜单层次结构




## 菜单控件

| Sr. No. | 控件& 描述 |
| --- | --- |
| 1 | JMenuBar JMenuBar的对象是与顶层窗口。 |
| 2 | JMenuItem 菜单中的项目必须属于JMenuItem的或任何其子类。 |
| 3 | JMenu JMenu对象是从菜单栏中显示一个下拉菜单组件。 |
| 4 | JCheckboxMenuItem JCheckBoxMenuItem 必须为 JMenuItem的子类。 |
| 5 | JRadioButtonMenuItem JRadioButtonMenuItem对象是JMenuItem的子类。 |
| 6 | JPopupMenu JPopupMenu弹出可以在一个组件内的指定位置动态弹出。 |

# TestNG教程

测试是检查应用程序的功能的过程是否按要求工作，以确保在开发层面，单元测试成图片。单元测试是单一实体（类或方法）的测试。单元测试是非常必要的，每一个软件公司向他们的客户提供高质量的产品。

JUnit 带动开发人员了解测试的实用性，尤其是单元测试的时候比任何其他测试框架。凭借一个相当简单，务实，严谨的架构，JUnit已经能够“感染”了一大批开发人员。 JUnit的特点，可以看看Junit 特点。

其中JUnit缺点：

- 最初的设计，使用于单元测试，现在只用于各种测试
- 不能依赖测试
- 配置控制欠佳（安装/拆卸）
- 侵入性（强制扩展类，并以某种方式命名方法）
- 静态编程模型（不必要的重新编译）
- 不同的适合管理复杂项目中的测试可以是非常棘手.

## TestNG是什么?

TestNG按照其文档的定义是：

TestNG是一个测试框架，其灵感来自JUnit和NUnit的，但引入了一些新的功能，使其功能更强大，使用更方便。

TestNG是一个开源自动化测试框架;TestNG表示下一代。 TestNG是类似于JUnit（特别是JUnit 4），但它不是一个JUnit扩展。它的灵感来源于JUnit。它的目的是优于JUnit的，尤其是当测试集成的类。 TestNG的创造者是Cedric Beust（塞德里克·博伊斯特）

TestNG消除了大部分的旧框架的限制，使开发人员能够编写更加灵活和强大的测试。 因为它在很大程度上借鉴了Java注解（JDK5.0引入的）来定义的测试，它也可以告诉你如何使用这个新功能在真实的Java语言生产环境中。

## TestNG的特点

- 注解
- TestNG使用Java和面向对象的功能

- 支持综合类测试（例如，默认情况下，没有必要创建一个新的测试每个测试方法的类的实例）
- 独立的编译时间测试代码运行时配置/数据信息
- 灵活的运行时配置
- 主要介绍“测试组”。当编译测试，只要问TestNG运行所有的“前端”的测试，或“快”，“慢”，“数据库”等
- 支持依赖测试方法，并行测试，负载测试，局部故障
- 灵活的插件API
- 支持多线程测试

# TestNG介绍 - TestNG教程

测试是检查应用程序的功能的过程是否按要求工作，以确保在开发层面，单元测试成图片。单元测试是单一实体（类或方法）的测试。单元测试是非常必要的，每一个软件公司向他们的客户提供高质量的产品。

JUnit 带动开发人员了解测试的实用性，尤其是单元测试的时候比任何其他测试框架。凭借一个相当简单，务实，严谨的架构，JUnit已经能够“感染”了一大批开发人员。 JUnit的特点，可以看看Junit 特点。

其中JUnit缺点：

- 最初的设计，使用于单元测试，现在只用于各种测试
- 不能依赖测试
- 配置控制欠佳（安装/拆卸）
- 侵入性（强制扩展类，并以某种方式命名方法）
- 静态编程模型（不必要的重新编译）
- 不同的适合管理复杂项目中的测试可以是非常棘手.

## TestNG是什么?

TestNG按照其文档的定义是：

TestNG是一个测试框架，其灵感来自JUnit和NUnit的，但引入了一些新的功能，使其功能更强大，使用更方便。

TestNG是一个开源自动化测试框架;TestNG表示下一代。 TestNG是类似于JUnit（特别是JUnit 4），但它不是一个JUnit扩展。它的灵感来源于JUnit。它的目的是优于JUnit的，尤其是当测试集成的类。 TestNG的创造者是Cedric Beust（塞德里克·博伊斯特）

TestNG消除了大部分的旧框架的限制，使开发人员能够编写更加灵活和强大的测试。 因为它在很大程度上借鉴了Java注解（JDK5.0引入的）来定义的测试，它也可以告诉你如何使用这个新功能在真实的Java语言生产环境中。

## TestNG的特点

- 注解
- TestNG使用Java和面向对象的功能

- 支持综合类测试（例如，默认情况下，没有必要创建一个新的测试每个测试方法的类的实例）
- 独立的编译时间测试代码运行时配置/数据信息
- 灵活的运行时配置
- 主要介绍“测试组”。当编译测试，只要问TestNG运行所有的“前端”的测试，或“快”，“慢”，“数据库”等
- 支持依赖测试方法，并行测试，负载测试，局部故障
- 灵活的插件API
- 支持多线程测试

# TestNG环境设置（配置安装） - TestNG教程

TestNG是一个Java的框架，所以第一个要求是JDK要安装在你的机器上。

## 系统要求

| JDK | 1.5或以上 |
|---|---|
| 内存 | 没有最低要求 |
| 磁盘空间 | 没有最低要求 |
| 操作系统 | 没有最低要求 |

## 步骤1 -验证Java安装在你的机器上

现在，打开控制台并执行以下的java命令。

| OS | 任务 | 命令 |
|---|---|---|
| Windows | 打开命令控制台 | c:> java -version |
| Linux | 打开命令终端 | $ java -version |
| Mac | 打开命令终端 | machine:~ joseph$ java -version |

让我们来验证所有的操作系统的输出：

Windows

```
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode) |
```

Linux

```
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode) |
```

Mac

```
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode) |
```

如果你没有安装Java，安装Java软件开发工具包（SDK）点击：http://www.oracle.com/technetwork/java/javase/downloads/index.html. 我们假设本教程中安装和使用Java1.7.0_25版本。

## 第二步：设置JAVA环境

设置JAVA_HOME环境变量指向的基本目录的位置，在你的机器上安装Java。例如：

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 JAVA_HOME 为 C:Program FilesJavajdk1.7.0_25 |
| Linux | export JAVA_HOME=/usr/local/java-current |
| Mac | export JAVA_HOME=/Library/Java/Home |

添加Java编译器的位置，系统路径。

| OS | 输出 |
|---|---|
| Windows | Append the string; C:Program FilesJavajdk1.7.0_25in to the end of the system variable, Path. |
| Linux | export PATH=$PATH:$JAVA_HOME/bin/ |
| Mac | not required |

验证Java安装使用命令java-version如上所述。

## 第3步：下载TestNG的归档文件

下载最新版本的TestNG的jar文件，详细请点击访问 http://www.testng.org.。在写这篇教程的时候，我下载TestNG中-6.8.jar，并将 testng-6.8.jar 其复制到 C:>TestNG 目录。

| OS | 压缩文件名 |
|---|---|
| Windows | testng-6.8.jar |
| Linux | testng-6.8.jar |
| Mac | testng-6.8.jar |

# 步骤4：设置TestNG的环境

设置TESTNG_HOME环境变量指向TestNG的jar 存放在您的机器上的基本目录位置。假设，我们已经储存了testng-6.8.jar， TestNG各种操作系统上的文件夹如下：

| OS | 输出 |
|---|---|
| Windows | Set the environment variable TESTNG_HOME to C:TESTNG |
| Linux | export TESTNG_HOME=/usr/local/TESTNG |
| Mac | export TESTNG_HOME=/Library/TESTNG |

# 第5步：设置CLASSPATH变量

设置CLASSPATH环境变量指向TestNG的jar文件位置。假设，我们已经储存了testng-6.8.jar, TestNG在各种操作系统上的文件夹如下：

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 CLASSPATH 为 %CLASSPATH%;%TESTNG_HOME% estng-6.8.jar; |
| Linux | export CLASSPATH=$CLASSPATH:$TESTNG_HOME/testng-6.8.jar: |
| Mac | export CLASSPATH=$CLASSPATH:$TESTNG_HOME/testng-6.8.jar: |

# 步骤6：测试TestNG的设置

创建一个Java类文件名TestNGSimpleTest C: > TestNG_WORKSPACE

```
import org.testng.annotations.Test;
import static org.testng.Assert.assertEquals;

public class TestNGSimpleTest {
    @Test
    public void testAdd() {
        String str = "TestNG is working fine";
        assertEquals("TestNG is working fine", str);
    }
}
```

TestNG的几种不同的方法可以被调用：

- testng.xml 文件
- ant
- 命令行

让我们调用使用testng.xml文件。创建一个XML文件名称testng.xml C: > TestNG_WORKSPACE 执行测试用例(s)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
  <test name="test1">
    <classes>
       <class name="TestNGSimpleTest"/>
    </classes>
  </test>
</suite>
```

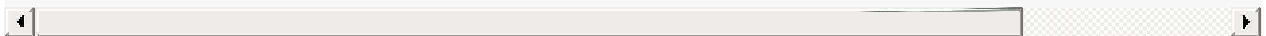
## 第7步：检查结果

类编译使用javac编译如下：

```
C:TestNG_WORKSPACE>javac TestNGSimpleTest.java
```

现在，调用testng.xml看到的结果：

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```



验证输出

```
===============================================
Suite1
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

# TestNG编写测试 - TestNG教程

编写TestNG测试基本上包括以下步骤：

- 测试和编写业务逻辑，在代码中插入TestNG的注解..
- 添加一个testng.xml文件或build.xml中在测试信息（例如类名，您想要运行的组，等..）
- 运行 TestNG.

在这里，我们将看到一个完整的例子了TestNG测试使用POJO类，业务逻辑类，将通过TestNG的运行测试XML。

创建 EmployeeDetails.java 在 C: > TestNG_WORKSPACE 是一个 POJO 类.

```
public class EmployeeDetails {

   private String name;
   private double monthlySalary;
   private int age;

   /**
   * @return the name
   */
   public String getName() {
      return name;
   }
   /**
   * @param name the name to set
   */
   public void setName(String name) {
      this.name = name;
   }
   /**
   * @return the monthlySalary
   */
   public double getMonthlySalary() {
      return monthlySalary;
   }
   /**
   * @param monthlySalary the monthlySalary to set
   */
   public void setMonthlySalary(double monthlySalary) {
      this.monthlySalary = monthlySalary;
   }
   /**
   * @return the age
   */
   public int getAge() {
      return age;
   }
   /**
   * @param age the age to set
   */
   public void setAge(int age) {
   this.age = age;
   }
}
```

EmployeeDetails 类是用来

- get/set 员工的名字的值
- get/set 员工月薪的值
- get/set员工年龄的值

创建一个 EmpBusinessLogic.java 在 C: > TestNG_WORKSPACE 其中包含业务逻辑

```
public class EmpBusinessLogic {
   // Calculate the yearly salary of employee
   public double calculateYearlySalary(EmployeeDetails employeeDeta
      double yearlySalary=0;
      yearlySalary = employeeDetails.getMonthlySalary() * 12;
      return yearlySalary;
   }

   // Calculate the appraisal amount of employee
   public double calculateAppraisal(EmployeeDetails employeeDetails
      double appraisal=0;
      if(employeeDetails.getMonthlySalary() < 10000){
         appraisal = 500;
      }else{
         appraisal = 1000;
      }
      return appraisal;
   }
}
```

EmpBusinessLogic 类用于计算

- 员工的年薪
- 考核支付予雇员

现在，让我们创建一个TestNG 类名称为 TestEmployeeDetails.java 在 C: > TestNG_WORKSPACE. TestNG类是一个Java类，它包含至少一个TestNG的注解。 这个类包含测试用例进行测试。可以配置，@BeforeXXX和@AfterXXX注解了TestNG测试 (在本章，我们会看到这样TestNG - Execution Procedure) 它允许执行一些Java逻辑的目标点之前和之后。

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class TestEmployeeDetails {
    EmpBusinessLogic empBusinessLogic = new EmpBusinessLogic();
    EmployeeDetails employee = new EmployeeDetails();

    @Test
    public void testCalculateAppriasal() {
        employee.setName("Rajeev");
        employee.setAge(25);
        employee.setMonthlySalary(8000);
        double appraisal = empBusinessLogic
            .calculateAppraisal(employee);
        Assert.assertEquals(500, appraisal, 0.0, "500");
    }

    // test to check yearly salary
    @Test
    public void testCalculateYearlySalary() {
        employee.setName("Rajeev");
        employee.setAge(25);
        employee.setMonthlySalary(8000);
        double salary = empBusinessLogic
            .calculateYearlySalary(employee);
        Assert.assertEquals(96000, salary, 0.0, "8000");
    }
}
```

TestEmployeeDetails 测试方法，用于类EmpBusinessLogic，它

- 雇员测试的年薪
- 测试评估员工的金额

之前，你可以运行测试，但是必须使用特殊的XML文件，通常命名为testng.xml配置TestNG。此文件的语法很简单，其内容如下。创建这个文件C: > TestNG_WORKSPACE:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
  <test name="test1">
    <classes>
       <class name="TestEmployeeDetails"/>
    </classes>
  </test>
</suite>
```

以上文件的详情如下：

- suite代表一个XML文件。它可以包含一个或多个测试，并被定义由<suite>标记
- 标签<test>代表一个测试，并可以包含一个或多个TestNG的类
- <class>的标签代表一个TestNG的类是一个Java类，它包含至少一个TestNG的注解。它可以包含一个或多个测试方法。

编译使用javac测试用例类。

```
C:TestNG_WORKSPACE>javac EmployeeDetails.java EmpBusinessLogic.java
```

现在TestNG用下面的命令：

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

如果所有配置正确的话，你应该看到测试结果，在控制台中。此外，TestNG创建了一个非常漂亮的HTML报告，会自动在当前目录下创建一个文件夹名为test-output。如果打开并加载的index.html，会看到类似下面的图片中的一个页面：

**test1**

| | |
|---|---|
| Tests passed/Failed/Skipped: | 2/0/0 |
| Started on: | Mon Aug 12 19:40:08 IST 2013 |
| Total time: | 0 seconds (15 ms) |
| Included groups: | |
| Excluded groups: | |

(Hover the method name to see the test class name)

| PASSED TESTS | | | |
|---|---|---|---|
| Test method | Exception | Time (seconds) | Instance |
| testCalculateAppriasal<br>Test class: TestEmployeeDetails | | 0 | TestEmployeeDetails@6c3c130a |
| testCalculateYearlySalary<br>Test class: TestEmployeeDetails | | 0 | TestEmployeeDetails@6c3 |

# TestNG基本注解(注释) - TestNG教程

传统的方式来表示JUnit 3中的测试方法是测试自己的名字前缀。标记一个类中的某些方法，具有特殊的意义，这是一个非常有效的方法，但命名不很好的扩展（如果我们想添加更多标签为不同的框架？），而非缺乏灵活性（如果我们要通过额外的参数测试框架）。

注释被正式加入到JDK 5中的Java语言和TestNG作出选择使用注释注释测试类。

这里是TestNG的支持列表中的注解：

| 注解 | 描述 |
|---|---|
| **@BeforeSuite** | 注解的方法将只运行一次，运行所有测试前此套件中。 |
| **@AfterSuite** | 注解的方法将只运行一次此套件中的所有测试都运行之后。 |
| **@BeforeClass** | 注解的方法将只运行一次先行先试在当前类中的方法调用。 |
| **@AfterClass** | 注解的方法将只运行一次后已经运行在当前类中的所有测试方法。 |
| **@BeforeTest** | 注解的方法将被运行之前的任何测试方法属于内部类的<test>标签的运行。 |
| **@AfterTest** | 注解的方法将被运行后，所有的测试方法，属于内部类的<test>标签的运行。 |
| **@BeforeGroups** | 组的列表，这种配置方法将之前运行。此方法是保证在运行属于任何这些组第一个测试方法，该方法被调用。 |
| **@AfterGroups** | 组的名单，这种配置方法后，将运行。此方法是保证运行后不久，最后的测试方法，该方法属于任何这些组被调用。 |
| **@BeforeMethod** | 注解的方法将每个测试方法之前运行。 |
| **@AfterMethod** | 被注释的方法将被运行后，每个测试方法。 |
| **@DataProvider** | 标志着一个方法，提供数据的一个测试方法。注解的方法必须返回一个Object[] []，其中每个对象[]的测试方法的参数列表中可以分配。该@Test 方法，希望从这个DataProvider的接收数据，需要使用一个dataProvider名称等于这个注解的名字。 |
| **@Factory** | 作为一个工厂，返回TestNG的测试类的对象将被用于标记的方法。该方法必须返回Object[]。 |
| **@Listeners** | 定义一个测试类的监听器。 |
| **@Parameters** | 介绍如何将参数传递给@Test方法。 |
| **@Test** | 标记一个类或方法作为测试的一部分。 |

# 使用注释的好处

以下是一些使用注释的好处：

- TestNG的标识的方法关心寻找注解。因此，方法名并不限于任何模式或格式。
- 我们可以通过额外的参数注解。
- 注释是强类型的，所以编译器将标记任何错误。

- 测试类不再需要任何东西（如测试案例，在JUnit3）扩展。

# TestNG执行程序 - TestNG教程

本教程介绍了TestNG中执行程序的方法，这意味着该方法被称为第一和一个接着。下面是执行程序的TestNG测试API的方法的例子。

创建一个Java类文件名TestngAnnotation.java在C:>TestNG_WORKSPACE测试注解。

```
import org.testng.annotations.Test;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.AfterSuite;

public class TestngAnnotation {
    // test case 1
    @Test
    public void testCase1() {
        System.out.println("in test case 1");
    }

    // test case 2
    @Test
    public void testCase2() {
        System.out.println("in test case 2");
    }

    @BeforeMethod
    public void beforeMethod() {
        System.out.println("in beforeMethod");
    }

    @AfterMethod
    public void afterMethod() {
        System.out.println("in afterMethod");
    }

    @BeforeClass
    public void beforeClass() {
        System.out.println("in beforeClass");
    }

    @AfterClass
    public void afterClass() {
        System.out.println("in afterClass");
```

```
    }

    @BeforeTest
    public void beforeTest() {
        System.out.println("in beforeTest");
    }

    @AfterTest
    public void afterTest() {
        System.out.println("in afterTest");
    }

    @BeforeSuite
    public void beforeSuite() {
        System.out.println("in beforeSuite");
    }

    @AfterSuite
    public void afterSuite() {
        System.out.println("in afterSuite");
    }

}
```

接下来，让我们创建的文件 **testng.xml** 在 **C: > TestNG_WORKSPACE** 执行注解。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
  <test name="test1">
    <classes>
       <class name="TestngAnnotation"/>
    </classes>
  </test>
</suite>
```

编译使用javac测试用例类。

```
C:TestNG_WORKSPACE>javac TestngAnnotation.java
```

现在运行testng.xml，将运行提供的测试用例类中定义的测试用例。

```
C:TestNG_WORKSPACE>java org.testng.TestNG testng.xml
```

验证输出。

```
in beforeSuite
in beforeTest
in beforeClass
in beforeMethod
in test case 1
in afterMethod
in beforeMethod
in test case 2
in afterMethod
in afterClass
in afterTest
in afterSuite

===============================================
Suite
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

见上面的输出，TestNG是执行过程如下：

- 首先所有beforeSuite（）方法只执行一次。
- 最后，afterSuite的（）方法只执行一次。
- 即使方法 beforeTest(), beforeClass(), afterClass() 和afterTest() 方法只执行一次。
- beforeMethod（）方法执行每个测试用例，但在此之前执行的测试用例。
- afterMethod（）方法执行每个测试用例，但测试用例执行后。
- In between beforeMethod() and afterMethod() each test case executes.

# TestNG执行测试 - TestNG教程

使用TestNG类执行测试用例。这个类的主入口点在TestNG的框架运行测试。用户可以创建自己的TestNG的对象，并调用它以许多不同的方式：

- 在现有的testng.xml
- 合成testng.xml，完全从Java创建
- 直接设定测试类

您还可以定义哪些群体包括或排除，分配参数，命令行参数：

- -d outputdir: 指定输出目录
- -testclass class_name: 指定了一个或多个类名
- -testjar jar_name: 指定的jar包含测试
- -sourcedir src1;src2: ; 分隔源目录列表（只有当使用的javadoc注释）
- -target
- -groups
- -testrunfactory
- -listener

testng.xml现有在下面的例子中，我们将创建TestNG的对象。

## 创建一个类

- 创建一个Java类进行测试为 MessageUtil.java 在 C: > TestNG_WORKSPACE

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {

   private String message;

   //Constructor
   //@param message to be printed
   public MessageUtil(String message){
      this.message = message;
   }

   // prints the message
   public String printMessage(){
      System.out.println(message);
      return message;
   }
}
```

## 创建测试例类

- 创建一个Java测试类 SampleTest.java
- 您的测试类添加一个的测试方法testPrintMessage（）
- 添加注释@Test 到方法 testPrintMessage()
- 实现测试条件和使用的assertEquals API TestNG的检查条件

创建一个Java类文件名 SampleTest.java在 C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class SampleTest {

   String message = "Hello World";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test
   public void testPrintMessage() {
        Assert.assertEquals(message, messageUtil.printMessage());
   }
}
```

## 创建 **testng.xml**

接下来，让我们创建testng.xml文件在 C: > TestNG_WORKSPACE 执行测试用例，此文件捕获整个测试XML。这个文件可以很容易地描述所有的测试套件和它们的参数在一个文件中，你可以检查你的代码库或e-mail给同事。这也使得它容易提取测试或分裂的几个运行时配置的子集（例如，TestNG的database.xml 只能运行测试，行使数据库）。

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Sample test Suite">
   <test name="Sample test">
    <classes>
      <class name="SampleTest" />
    </classes>
  </test>
</suite>
```

情况下使用javac编译测试

```
C:TestNG_WORKSPACE>javac MessageUtil.java SampleTest.java
```

现在，运行这个 testng.xml，将运行中定义的测试用例 <test> 标签

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出。

```
Hello World

===============================================
Sample test Suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

# TestNG套件测试 - TestNG教程

## TestNG套件测试

测试套件的测试是为了测试软件程序的行为或一系列行为的情况下，是一个集合。在TestNG，我们不能定义一套测试源代码，但它代表的套件是一个XML文件执行特征。这也允许灵活的配置要运行的测试。套件可以包含一个或多个测试和被定义由<suite>标签。

testng.xml中有<suite>根标签。它描述了一个测试套件，这反过来又是由多个<test>区段组成。

下表列出了所有的<suite>可接受合法属性。

| 属性 | 描述 |
| --- | --- |
| name | 此套件的名称。这是一个强制性的属性。 |
| verbose | 这个运行级别或冗长。 |
| parallel | 由TestNG 运行不同的线程来运行此套件。 |
| thread-count | 使用的线程数，如果启用并行模式（忽略其他方式）。 |
| annotations | 在测试中使用注释的类型。 |
| time-out | 默认的超时时间，将用于本次测试中发现的所有测试方法。 |

在本章中，我们会告诉你一个例子，有两个Test1 & Test2测试类一起运行测试套件。

## 创建一个类

创建一个Java类进行测试 MessageUtil.java 在 C: > JUNIT_WORKSPACE

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {
    private String message;

    // Constructor
    // @param message to be printed
    public MessageUtil(String message) {
        this.message = message;
    }

    // prints the message
    public String printMessage() {
        System.out.println(message);
        return message;
    }

    // add "Hi!" to the message
    public String salutationMessage() {
        message = "Hi!" + message;
        System.out.println(message);
        return message;
    }
}
```

## 创建测试用例类

创建一个Java类文件名 Test1.java 在C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class Test1 {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
    Assert.assertEquals(message, messageUtil.printMessage());
    }
}
```

创建一个Java类文件名 Test2.java 在C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class Test2 {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
        message = "Hi!" + "Manisha";
        Assert.assertEquals(message,messageUtil.salutationMessage()
    }
}
```

现在，让我们编辑写入testng.xml 在C: > TestNG_WORKSPACE，将包含<suite>标签如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
  <test name="exampletest1">
    <classes>
       <class name="Test1" />
    </classes>
  </test>
  <test name="exampletest2">
    <classes>
       <class name="Test2" />
    </classes>
  </test>
</suite>
```

Suite1 包括 exampletest1 和 exampletest2.

所有Java类编译使用javac。

```
C:TestNG_WORKSPACE>javac MessageUtil.java Test1.java Test2.java
```

现在运行 testng.xml，将运行提供的测试用例类中定义的测试用例。

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出。

```
Inside testPrintMessage()
Manisha
Inside testSalutationMessage()
Hi!Manisha

===============================================
Suite1
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

您也可以检查测试输出文件夹;下Suite1文件夹中，可以看到两个HTML创建的 exampletest1.html 和 exampletest2.html 内容如下：

**exampletest1**

| | |
|---|---|
| Tests passed/Failed/Skipped: | 1/0/0 |
| Started on: | Wed Aug 14 19:17:02 IST 2013 |
| Total time: | 0 seconds (15 ms) |
| Included groups: | |
| Excluded groups: | |

*(Hover the method name to see the test class name)*

| PASSED TESTS | | | |
|---|---|---|---|
| Test method | Exception | Time (seconds) | Instance |
| testPrintMessage<br>Test class: Test1 | | 0 | [illegible] |

Yiibai.com

**exampletest2**

| | |
|---|---|
| Tests passed/Failed/Skipped: | 1/0/0 |
| Started on: | Wed Aug 14 19:17:02 IST 2013 |
| Total time: | 0 seconds (1 ms) |
| Included groups: | |
| Excluded groups: | |

*(Hover the method name to see the test class name)*

| PASSED TESTS | | | |
|---|---|---|---|
| Test method | Exception | Time (seconds) | Instance |
| testSalutationMessage<br>Test class: Test2 | | 0 | [illegible] |

Yiibai.com

# TestNG忽略测试 - TestNG教程

有时，我们的代码是没有准备好，如果测试用例写入到测试方法/代码将无法运行，在这种情况下，@Test(enabled = false)有助于禁用此测试案例。

测试方法是标注了@Test(enabled = false)，那么并不是已经准备好测试的测试用例是绕过。

现在，让我们来看看测试@Test(enabled = false) 动作。

## 创建一个类

- 创建一个Java类进行测试为 MessageUtil.java 在 C: > TestNG_WORKSPACE

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {

   private String message;

   //Constructor
   //@param message to be printed
   public MessageUtil(String message){
      this.message = message;
   }

   // prints the message
   public String printMessage(){
      System.out.println(message);
      return message;
   }

   // add "Hi!" to the message
   public String salutationMessage(){
      message = "Hi!" + message;
      System.out.println(message);
      return message;
   }
}
```

## 创建测试案例类

- 创建Java测试类为 IgnoreTest.java.
- 测试类添加测试方法testPrintMessage()，testSalutationMessage()。

- 添加注释 @Test(enabled = false) 到方法 testPrintMessage().

创建一个Java类文件名 IgnoreTest.java 在 C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class IgnoreTest {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test(enabled = false)
    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
        message = "Manisha";
    Assert.assertEquals(message, messageUtil.printMessage());
    }

    @Test
    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
    message = "Hi!" + "Manisha";
    Assert.assertEquals(message, messageUtil.salutationMessage());
    }
}
```

# 创建 testng.xml

创建一个文件 testng.xml C: > TestNG_WORKSPACE 用来执行测试案例

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http
<suite name="Suite1">
  <test name="test1">
    <classes>
       <class name="IgnoreTest" />
    </classes>
  </test>
 </suite>
```

编译MessageUtil的测试用例类使用javac。

```
C:TestNG_WORKSPACE>javac MessageUtil.java IgnoreTest.java
```

现在，运行testng.xml，将无法运行testPrintMessage（）定义的测试用例在测试案例类。

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出。 testPrintMessage（）测试用例没有测试。

```
Inside testSalutationMessage()
Hi!Manisha

===============================================
Suite1
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

也可以忽略一组测试将在下一章中讨论

# TestNG组测试 - TestNG教程

在TestNG中组测试是一个新的创新功能，它不存在于JUnit框架，它允许调度到适当的部分方法和瓶坯复杂的测试方法分组。您不仅可以声明属于群体的那些方法，但你也可以指定一组包含其他组。然后，TestNG可调用和要求包括一组特定的群体（或正则表达式），而排除另一个集合。这给了你最大的灵活性，如何分区测试，如果想运行两套不同的测试背靠背，不要求重新编译任何东西。

组指定testng.xml文件使用<groups>标签。它可以发现无论是根据<test>或<suite>标签。组指定<suite>标签适用于所有的的<test>标签下方。

现在，让我们看一个例子，如何组测试。

## 创建一个类

- 创建一个Java类进行测试为 MessageUtil.java 在 C: > TestNG_WORKSPACE

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {
    private String message;

    // Constructor
    // @param message to be printed
    public MessageUtil(String message) {
        this.message = message;
    }

    // prints the message
    public String printMessage() {
        System.out.println(message);
    return message;
    }

    // add "tutorialspoint" to the message
    public String salutationMessage() {
        message = "tutorialspoint" + message;
    System.out.println(message);
    return message;
    }

    // add "www." to the message
    public String exitMessage() {
    message = "www." + message;
    System.out.println(message);
    return message;
    }
}
```

## 创建测试案例类

- 创建一个Java测试类为 GroupTestExample.java.
- 测试类添加测试方法testPrintMessage（）和 testSalutationMessage（）。
- 组的测试方法两个类别为：
    - 检入登记测试（checkintest）：提交新的代码之前，你应该运行这些测试。他们通常应快，只要确保没有被打破的基本功能。
    - 功能测试（functest）：这些测试应该涵盖软件的所有功能，每天至少运行一次，虽然理想情况下，会希望他们不断运行。

创建Java类文件名 GroupTestExample.java 在 C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class GroupTestExample {
    String message = ".com";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test(groups = { "functest", "checkintest" })
    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
    message = ".com";
    Assert.assertEquals(message, messageUtil.printMessage());
    }

    @Test(groups = { "checkintest" })
    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
    message = "tutorialspoint" + ".com";
    Assert.assertEquals(message, messageUtil.salutationMessage());
    }

    @Test(groups = { "functest" })
    public void testingExitMessage() {
        System.out.println("Inside testExitMessage()");
        message = "www." + "tutorialspoint"+".com";
    Assert.assertEquals(message, messageUtil.exitMessage());
    }
}
```

## 创建testng.xml

创建一个文件 testng.xml C: > TestNG_WORKSPACE 来执行测试用例，在这里，我们将只执行这些测试，属于组functest。

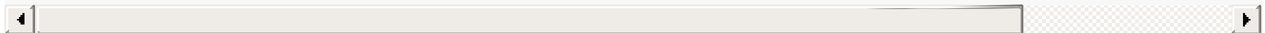
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
    <test name="test1">
        <groups>
        <run>
        <include name="functest" />
        </run>
    </groups>
    <classes>
        <class name="GroupTestExample" />
    </classes>
    </test>
</suite>
```

编译MessageUtil的测试用例类使用javac。

```
C:TestNG_WORKSPACE>javac MessageUtil.java GroupTestExample.java
```

现在，运行testng.xml，只运行的方法testPrintMessage（），因为它属于组functest。

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```



验证输出。只有的方法testPrintMessage（）被执行。

```
Inside testPrintMessage()
.com
Inside testExitMessage()
www..com

===============================================
Suite1
Total tests run: 2, Failures: 1, Skips: 0
===============================================
```

## 组中组

组也可以包含其他组。这些组称为MetaGroups。例如，您可能希望定义一个组中的所有，包括checkintest和functest。让我们修改testng.xml文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
   <test name="test1">
      <groups>
         <define name="all">
        <include name="functest"/>
        <include name="checkintest"/>
     </define>
     <run>
        <include name="all"/>
     </run>
        </groups>
     <classes>
          <class name="GroupTestExample" />
    </classes>
   </test>
</suite>
```

执行上述testng.xml将执行所有三个测试会给你下面的结果：

```
Inside testPrintMessage()
.com
Inside testSalutationMessage()
tutorialspoint.com
Inside testExitMessage()
www.tutorialspoint.com

===============================================
Suite1
Total tests run: 3, Failures: 0, Skips: 0
===============================================
```

# 排斥组

可以忽略一个组使用<exclude>标签，如下图所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
   <test name="test1">
      <groups>
         <define name="all">
        <exclude name="functest"/>
        <include name="checkintest"/>
     </define>
     <run>
        <include name="all"/>
     </run>
        </groups>
     <classes>
          <class name="GroupTestExample" />
    </classes>
   </test>
</suite>
```

# TestNG异常测试 - TestNG教程

TestNG跟踪异常处理代码提供了一个选项。可以测试是否需要代码抛出异常或不抛出。 @Test注释expectedExceptions 参数一起使用。现在，让我们来看看@Test（expectedExceptions）在动作中。

## 创建一个类

- 创建一个Java类进行测试说MessageUtil.java 在 C: > TestNG_WORKSPACE
- 在printMessage（）方法里添加一个错误条件

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {

   private String message;

   //Constructor
   //@param message to be printed
   public MessageUtil(String message){
      this.message = message;
   }

   // prints the message
   public void printMessage(){
      System.out.println(message);
      int a =0;
      int b = 1/a;
   }

   // add "Hi!" to the message
   public String salutationMessage(){
      message = "Hi!" + message;
      System.out.println(message);
      return message;
   }
}
```

## 创建测试案例类

- 创建一个Java测试类为 ExpectedExceptionTest.java。
- 添加的ArithmeticException 和 testPrintMessage（）测试用例的预期异常。

创建一个Java类文件名ExpectedExceptionTest.java 在 C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class ExpectedExceptionTest {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test(expectedExceptions = ArithmeticException.class)
    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
        messageUtil.printMessage();
   }
   @Test
   public void testSalutationMessage() {
      System.out.println("Inside testSalutationMessage()");
      message = "Hi!" + "Manisha";
      Assert.assertEquals(message,messageUtil.salutationMessage());
   }
}
```

## 创建测试运行

创建 testng.xml 在 C: > TestNG_WORKSPACE 执行测试案例。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
   <test name="test1">
           <classes>
          <class name="ExpectedExceptionTest" />
    </classes>
   </test>
</suite>
```

编译MessageUtil 测试用例类使用javac

```
C:TestNG_WORKSPACE>javac MessageUtil.java TestJunit.java
```

现在，运行测试运行，这将运行提供的测试用例类中定义的测试用例。

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出。testPrintMessage（）测试的情况下会获得通过。

```
Inside testPrintMessage()
Manisha
Inside testSalutationMessage()
Hi!Manisha

===============================================
Suite1
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

# TestNG依赖测试 - TestNG教程

有时候，你可能需要在一个特定的顺序调用方法在测试案例，或你想分享一些数据和方法之间的状态。TestNG支持这种依赖测试方法之间的显式依赖它支持声明。

TestNG允许指定依赖，无论与否：

- 使用属性dependsOnMethods在 @Test 注释OR
- 使用属性dependsOnGroups在@Test注解。

## 使用属性dependsOnMethods例如

### 创建一个类

创建一个Java类进行测试为 MessageUtil.java 在 C: > TestNG_WORKSPACE

```
public class MessageUtil {
    private String message;

    // Constructor
    // @param message to be printed
    public MessageUtil(String message) {
        this.message = message;
    }

    // prints the message
    public String printMessage() {
        System.out.println(message);
    return message;
    }

    // add "Hi!" to the message
    public String salutationMessage() {
    message = "Hi!" + message;
    System.out.println(message);
    return message;
    }
}
```

### 创建测试案例类

- 创建一个Java测试类为 DependencyTestUsingAnnotation.java.
- 添加方法 testPrintMessage(), testSalutationMessage() 和 initEnvironmentTest() 到测试类

- 添加属性 dependsOnMethods = { "initEnvironmentTest" } to the @Test 注释 oftestSalutationMessage() 方法.

创建Java类文件名 DependencyTestUsingAnnotation.java 在 C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class DependencyTestUsingAnnotation {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testPrintMessage() {
    System.out.println("Inside testPrintMessage()");
    message = "Manisha";
    Assert.assertEquals(message, messageUtil.printMessage());
    }

    @Test(dependsOnMethods = { "initEnvironmentTest" })
    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
    message = "Hi!" + "Manisha";
    Assert.assertEquals(message, messageUtil.salutationMessage());
    }

    @Test
    public void initEnvironmentTest() {
    System.out.println("This is initEnvironmentTest");
    }
}
```

## 创建TESTNG.XML

创建一个文件 testng.xml 在 C: > TestNG_WORKSPACE 来执行测试用例

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http
<suite name="Suite1">
    <test name="test1">
    <classes>
        <class name="DependencyTestUsingAnnotation" />
    </classes>
    </test>
</suite>
```

编译MessageUtil的测试用例类使用javac

```
C:TestNG_WORKSPACE>javac MessageUtil.java DependencyTestUsingAnnota
```

现在运行 testng.xml 这将会运行 testSalutationMessage() 只有在执行 ofinitEnvironmentTest() 方法之后

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出

```
This is initEnvironmentTest
Inside testPrintMessage()
Manisha
Inside testSalutationMessage()
Hi!Manisha

===============================================
Suite1
Total tests run: 3, Failures: 0, Skips: 0
===============================================
```

# 示例，使用属性dependsOnGroups

也可以依赖于整个群组的方法。让我们来看看下面的例子：

## 创建一个类

创建一个Java类进行测试为 MessageUtil.java 在 C: > TestNG_WORKSPACE

```
public class MessageUtil {
    private String message;

    // Constructor
    // @param message to be printed
    public MessageUtil(String message) {
        this.message = message;
    }

    // prints the message
    public String printMessage() {
        System.out.println(message);
    return message;
    }

    // add "Hi!" to the message
    public String salutationMessage() {
    message = "Hi!" + message;
    System.out.println(message);
    return message;
    }
}
```

## 创建测试案例类

- 创建一个Java测试类说依赖TestUsingAnnotation.java.
- 添加测试方法 testPrintMessage(), testSalutationMessage() 和 initEnvironmentTest() 测试类和他们的组 "初始化"
- 添加属性 dependsOnMethods = { "init.*" } to the @Test 注释 testSalutationMessage() 方法

创建Java类文件名 DependencyTestUsingAnnotation.java 在 C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class DependencyTestUsingAnnotation {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test(groups = { "init" })
    public void testPrintMessage() {
    System.out.println("Inside testPrintMessage()");
    message = "Manisha";
    Assert.assertEquals(message, messageUtil.printMessage());
    }

    @Test(dependsOnGroups = { "init.*" })
    public void testSalutationMessage() {
    System.out.println("Inside testSalutationMessage()");
    message = "Hi!" + "Manisha";
    Assert.assertEquals(message, messageUtil.salutationMessage());
    }

    @Test(groups = { "init" })
    public void initEnvironmentTest() {
    System.out.println("This is initEnvironmentTest");
    }
}
```

在这个例子中，testSalutationMessage（）被声明为根据任何一组匹配正则表达式“的init*”，这保证了，一种方法，testPrintMessage的（）和initEnvironmentTest（）将始终前testSalutationMessage（）被调用。

> 如果一个方法失败，取决于你有一个很难依赖于它（alwaysRun= false，这是默认的），没有标记的方法依赖于它的失败，但作为跳过。跳过的方法将被报告为例如在最终报告（在HTML中，既不是红也不是绿的颜色），这是很重要的，因为跳过的方法不一定是失败。

## 创建TESTNG.XML

创建一个文件testng.xml C: > TestNG_WORKSPACE 执行测试案例

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http
<suite name="Suite1">
    <test name="test1">
    <classes>
        <class name="DependencyTestUsingAnnotation" />
    </classes>
    </test>
</suite>
```

编译MessageUtil的测试用例类使用javac

```
C:TestNG_WORKSPACE>javac MessageUtil.java DependencyTestUsingAnnota
```

现在，运行testng.xml，这将运行testSalutationMessage（）方法后，才执行initEnvironmentTest（）方法。

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出

```
This is initEnvironmentTest
Inside testPrintMessage()
Manisha
Inside testSalutationMessage()
Hi!Manisha

===============================================
Suite1
Total tests run: 3, Failures: 0, Skips: 0
===============================================
```

## dependsOnGroups Vs dependsOnMethods

- 在使用组，我们不再面临重构的问题。只要我们不修改dependsOnGroups或组属性，我们的测试将继续运行，设立适当的依赖。
- 每当一个新的方法需要添加依存关系图中，我们需要做的就是把它正确的组中，并确保它依赖于正确的组。我们不需要修改任何其他方法。

# TestNG参数化测试 - TestNG教程

在TestNG的另一个有趣的功能是参数测试。在大多数情况下，你会遇到这样一个场景，业务逻辑需要一个巨大的不同数量的测试。参数测试，允许开发人员运行同样的测试，一遍又一遍使用不同的值。

TestNG让你直接传递参数测试方法两种不同的方式：

- 使用testng.xml
- 数据提供程序

## 传递参数使用testng.xml

有了这种技术，在testng.xml文件中定义的简单参数，然后在源文件中引用这些参数。让我们看看下面的例子中如何使用这种技术来传递参数。

### 创建测试案例类

- 创建一个Java测试类 ParameterizedTest1.java.
- 测试方法parameterTest（）添加到测试类。此方法需要一个字符串作为输入参数。
- 添加注释 @Parameters("myName") 到此方法。该参数将被传递testng.xml，在下一步我们将看到一个值。

创建Java类文件名 ParameterizedTest1.java 在 C: > TestNG_WORKSPACE

```
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class ParameterizedTest1 {
    @Test
    @Parameters("myName")
    public void parameterTest(String myName) {
        System.out.println("Parameterized value is : " + myName);
    }
}
```

### 创建 TESTNG.XML

创建 testng.xml C: > TestNG_WORKSPACE 执行测试案例

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http
<suite name="Suite1">
    <test name="test1">
    <parameter name="myName" value="manisha"/>
    <classes>
        <class name="ParameterizedTest1" />
        </classes>
    </test>
</suite>
```

> 我们还可以定义参数在<suite>级别。假设我们已经定义在两个<suite>和<test>级别myName，在这种情况下，常规的作用域规则适用。这意味着，任何类里面<test>标签将查看值参数定义在<test>，而testng.xml文件中的类的其余部分将看到定义在<suite>中值

编译使用javac的测试用例类。

```
C:TestNG_WORKSPACE>javac ParameterizedTest1.java
```

现在，运行testng.xml，其中将运行parameterTest方法。TestNG的将试图找到一个命名myName的第一<test>标签的参数，然后，如果它不能找到它，它会搜索包围在的<suit>标签。

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出。

```
Parameterized value is : manisha

===============================================
Suite1
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

TestNG 对testng.xml 的参数的类型指定的值会自动尝试转换。下面是支持的类型：

- String
- int/Integer
- boolean/Boolean
- byte/Byte

- char/Character
- double/Double
- float/Float
- long/Long
- short/Short

# 传递参数与数据提供者

当你需要通过复杂的参数或参数需要创建从Java（复杂的对象，对象读取属性文件或数据库等..），在这种情况下，可以将参数传递使用数据提供者。数据提供者@DataProvider的批注的方法。这个注解只有一个字符串属性：它的名字。如果不提供名称，数据提供者的名称会自动默认方法的名称。数据提供者返回一个对象数组。

让我们看看下面的例子使用数据提供者。第一个例子是@DataProvider的使用Vector，String或Integer 作为参数，第二个例子是关于@DataProvider 的使用对象作为参数。

## 实例 1

在这里 @DataProvider 通过整数和布尔参数。

## 创建Java类

创建一个java类PrimeNumberChecker.java。这个类检查，如果是素数。创建这个类在 C: > TestNG_WORKSPACE

```
public class PrimeNumberChecker {
    public Boolean validate(final Integer primeNumber) {
        for (int i = 2; i < (primeNumber / 2); i++) {
            if (primeNumber % i == 0) {
                return false;
             }
        }
        return true;
    }
}
```

## 创建测试案例类

- 创建一个Java测试类 ParamTestWithDataProvider1.java.

- 定义方法primeNumbers（），其定义为DataProvider 使用注释。此方法返回的对象数组的数组。
- 测试方法testPrimeNumberChecker（）添加到测试类中。此方法需要一个整数和布尔值作为输入参数。这个方法验证，如果传递的参数是一个素数。
- 添加注释 @Test(dataProvider = "test1") 到此方法。dataProvider的属性被映射到"test1".

创建Java类文件名ParamTestWithDataProvider1.java 在 C: > TestNG_WORKSPACE

```
import org.testng.Assert;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class ParamTestWithDataProvider1 {
    private PrimeNumberChecker primeNumberChecker;

    @BeforeMethod
    public void initialize() {
        primeNumberChecker = new PrimeNumberChecker();
    }

    @DataProvider(name = "test1")
    public static Object[][] primeNumbers() {
        return new Object[][] { { 2, true }, { 6, false }, { 19, tr
            { 22, false }, { 23, true } };
    }

    // This test will run 4 times since we have 5 parameters define
    @Test(dataProvider = "test1")
    public void testPrimeNumberChecker(Integer inputNumber,
        Boolean expectedResult) {
    System.out.println(inputNumber + " " + expectedResult);
    Assert.assertEquals(expectedResult,
            primeNumberChecker.validate(inputNumber));
    }
}
```

## 创建 TESTNG.XML

创建 testng.xml C: > TestNG_WORKSPACE 执行测试案例。

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http
<suite name="Suite1">
    <test name="test1">
    <classes>
        <class name="ParamTestWithDataProvider1" />
        </classes>
    </test>
</suite>
```

编译使用javac的测试用例类。

```
C:TestNG_WORKSPACE>.javac ParamTestWithDataProvider1.java PrimeNumb
```

运行testng.xml.

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出。

```
2 true
6 false
19 true
22 false
23 true

===============================================
Suite1
Total tests run: 5, Failures: 0, Skips: 0
===============================================
```

## 实例 2

在这里，@DataProvider 传递对象作为参数。

## 创建Java类

创建一个Java类 Bean.java, 对象带有 get/set 方法, 在 C: > TestNG_WORKSPACE.

```
public class Bean {
    private String val;
    private int i;
    public Bean(String val, int i){
        this.val=val;
        this.i=i;
    }
    public String getVal() {
    return val;
    }
    public void setVal(String val) {
    this.val = val;
    }
    public int getI() {
    return i;
    }
    public void setI(int i) {
    this.i = i;
    }
}
```

## 创建测试案例类

- 创建一个Java测试类 ParamTestWithDataProvider2.java.
- 定义方法primeNumbers（），其定义为DataProvider使用注释。此方法返回的对象数组的数组。
- 添加测试类中测试方法TestMethod（）。此方法需要对象的bean作为参数。
- 添加注释 @Test(dataProvider = "test1") 到此方法. dataProvider 属性被映射到 "test1".

创建Java类文件名 ParamTestWithDataProvider2.java 在 C: > TestNG_WORKSPACE

```
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class ParamTestWithDataProvider2 {
    @DataProvider(name = "test1")
    public static Object[][] primeNumbers() {
        return new Object[][] { { new Bean("hi I am the bean", 111)
    }

    @Test(dataProvider = "test1")
    public void testMethod(Bean myBean) {
        System.out.println(myBean.getVal() + " " + myBean.getI());
    }
}
```

## 创建 TESTNG.XML

创建一个文件 testng.xml C: > TestNG_WORKSPACE 来执行测试用例.

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http
<suite name="Suite1">
    <test name="test1">
    <classes>
        <class name="ParamTestWithDataProvider2" />
        </classes>
    </test>
</suite>
```

编译使用javac的测试用例类。

```
C:TestNG_WORKSPACE>javac ParamTestWithDataProvider2.java Bean.java
```

运行 testng.xml.

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE" org.testng.TestNG
```

验证输出。

```
hi I am the bean 111

===============================================
Suite1
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

# TestNG运行JUnit测试 - TestNG教程

现在，您已经了解了TestNG和它的各种测试，如果现在担心如何重构现有的JUnit代码，那就没有必要，使用TestNG提供了一种方法，从JUnit和TestNG按照自己的节奏。也可以使用TestNG执行现有JUnit测试用例。

TestNG可以自动识别和运行JUnit测试，所以你可以使用TestNG运行所有的测试，并编写新的测试使用TestNG。所有你必须做的就是把JUnit的库TestNG的类路径上，它可以发现并使用JUnit类，改变测试运行从JUnit和TestNG Ant中，然后运行TestNG的“mixed”模式。这种方式可以在同一个项目中所有的测试，即使是在同一个包中，并开始使用TestNG。这种方法还可以转换您现有的JUnit测试到TestNG。

让我们来看看下面的例子中，并尝试了上述功能：

## 创建JUnit测试用例类

创建一个Java类，这是一个JUnit测试类， TestJunit.java 在 C: > TestNG_WORKSPACE

```
import org.junit.Test;
import static org.testng.AssertJUnit.assertEquals;

public class TestJunit {
    @Test
    public void testAdd() {
        String str= "Junit testing using TestNG";
        assertEquals("Junit testing using TestNG",str);
    }
}
```

现在，让我们来编写 testng.xml 在 C: > TestNG_WORKSPACE 应该包涵 <suite> 标签如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Converted JUnit suite" >
    <test name="JUnitTests" junit="true">
        <classes>
            <class name="TestJunit" />
        </classes>
    </test>
</suite>
```

要执行JUnit测试用例定义属性 junit="true" 如上面的xml文件中. JUnit测试用例类TestJunit定义在类名。

JUnit 4中，TestNG将使用 org.junit.runner.JUnitCore 运行测试。

所有Java类编译使用javac。

```
C:TestNG_WORKSPACE>javac TestJunit.java
```

现在运行testng.xml，这将运行TestNG的JUnit测试用例。

```
C:TestNG_WORKSPACE>java -cp "C:TestNG_WORKSPACE:C:TestNG_WORKSPACE1
```

在这里，我已经放在了 junit-4.11.jar 在 C:TestNG_WORKSPACElibjunit-4.11.jar下面.

验证输出。

```
===============================================
Converted JUnit suite

Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

# TestNG测试结果报告 - TestNG教程

报告是任何测试的执行是最重要的部分，原因是它可以帮助用户了解执行测试，故障点和失败的原因的结果。记录，另一方面，重要的是要留意执行流程，或在任何故障的情况下进行调试。

TestNG默认情况下，会产生不同类型的测试执行报告。这包括HTML和XML报表输出。 TestNG的还允许用户自己写的报告，并用它使用TestNG。还有一个选项来写你自己的记录器，在运行时通过TestNG的通知。

主要有两种方法来生成报告使用TestNG：

- 监听器: 为了实现一个监听类，类有实现theorg.testng。 ITestListener接口。这些类在运行时通知了TestNG测试开始时，结束后，失败，跳过或传递。
- 记录器: 为了实现一个报表类，实现一个org.testng.IReporter接口。这些类一整套运行结束时调用。调用时，该对象包含整个测试运行的信息传递到这个类。

下表列出了不同的情况报告和记录的例子：

| | |
|---|---|
| 自定义日志 | 这个例子说明如何编写您自己的记录。 |
| 自定义记录器 | 这个例子说明了如何编写自己的记录器。 |
| HTML 和 XML 报告 | 这个例子说明了默认的HTML和XML报告TestNG产生。 |
| JUnit 报告 | 这个例子说明了TestNG的报告生成Junit的报告。 |

# TestNG插件与ANT - TestNG教程

在这个例子中，我们将演示如何使用ANT运行TestNG。让我们遵循的步骤：

## 步骤1：下载Apache Ant

下载 Apache Ant

| OS | 压缩文件名 |
|---|---|
| Windows | apache-ant-1.8.4-bin.zip |
| Linux | apache-ant-1.8.4-bin.tar.gz |
| Mac | apache-ant-1.8.4-bin.tar.gz |

## 步骤2：设置Ant环境

设置ANT_HOME环境变量指向参考基本目录的位置，ANT库存储在您的机器上。例如，我们已经存储了Ant库apache-ant-1.8.4，各种操作系统上的文件夹如下：

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 ANT_HOME to C:Program FilesApache Software Foundationapache-ant-1.8.4 |
| Linux | export ANT_HOME=/usr/local/apache-ant-1.8.4 |
| Mac | export ANT_HOME=/Library/apache-ant-1.8.4 |

附加的Ant编译系统路径位置，在不同的操作系统如下：

| OS | 输出 |
|---|---|
| Windows | 追加字符串;%ANT_HOMEin 系统变量的结尾 |
| Linux | export PATH=$PATH:$ANT_HOME/bin/ |
| Mac | not required |

## 第3步：下载TestNG

下载http://www.testng.org.

| OS | Archive name |
|---|---|
| Windows | testng-6.8.jar |
| Linux | testng-6.8.jar |
| Mac | testng-6.8.jar |

## 第4步：创建项目结构

- 创建文件夹 TestNGWithAnt 在C: > TestNG_WORKSPACE
- 创建文件夹 src 在 C: > TestNG_WORKSPACE > TestNGWithAnt
- 创建文件夹 test 在 C: > TestNG_WORKSPACE > TestNGWithAnt
- 创建文件夹 lib 在 C: > TestNG_WORKSPACE > TestNGWithAnt
- 创建MessageUtil 类在 C: > TestNG_WORKSPACE > TestNGWithAnt > src 文件夹.

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {

   private String message;

   //Constructor
   //@param message to be printed
   public MessageUtil(String message){
      this.message = message;
   }

   // prints the message
   public void printMessage(){
      System.out.println(message);
      return message;
   }

   // add "Hi!" to the message
   public String salutationMessage(){
      message = "Hi!" + message;
      System.out.println(message);
      return message;
   }
}
```

- 创建TestMessageUtil 类在 C: > TestNG_WORKSPACE > TestNGWithAnt > src 目录.

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class TestMessageUtil {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
        Assert.assertEquals(message,messageUtil.printMessage());
    }

    @Test
    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
        message = "Hi!" + "Manisha";
        Assert.assertEquals(message,messageUtil.salutationMessage()
    }
}
```

- 拷贝 testng-6.8.jar 到 C: > TestNG_WORKSPACE > TestNGWithAnt > lib 文件夹

# 创建 ANT build.xml

首先，我们需要定义TestNG的ant任务如下：

```
<taskdef name="testng" classname="org.testng.TestNGAntTask">
    <classpath>
      <pathelement location="lib/testng-6.8.jar"/>
    </classpath>
  </taskdef>
```

然后我们使用 <testng> TestNG的测试案例Ant来执行任务。

C: > TestNG_WORKSPACE > TestNGWithAnt > build.xml 内容如下：

```
<project name="TestNGTest" default="test" basedir=".">
<!-- Define <testng> task -->
  <taskdef name="testng" classname="org.testng.TestNGAntTask">
    <classpath>
      <pathelement location="lib/testng-6.8.jar"/>
    </classpath>
  </taskdef>
   <property name="testdir" location="test" />
   <property name="srcdir" location="src" />
   <property name="libdir" location="lib" />
   <property name="full-compile" value="true" />
   <path id="classpath.base"/>
   <path id="classpath.test">
       <fileset dir="${libdir}">
         <include name="**/*.jar" />
      </fileset>
      <pathelement location="${testdir}" />
      <pathelement location="${srcdir}" />
      <path refid="classpath.base" />
   </path>
   <target name="clean" >
      <delete verbose="${full-compile}">
         <fileset dir="${testdir}" includes="**/*.class" />
      </delete>
   </target>
   <target name="compile" depends="clean">
      <javac srcdir="${srcdir}" destdir="${testdir}"
         verbose="${full-compile}">
         <classpath refid="classpath.test"/>
      </javac>
   </target>
   <target name="test" depends="compile">
    <testng outputdir="${testdir}" classpathref="classpath.test">
      <xmlfileset dir="${srcdir}" includes="testng.xml"/>
    </testng>
   </target>
</project>
```

执行下列ant命令。

```
C:TestNG_WORKSPACETestNGWithAnt>ant
```

验证输出

```
test:
   [testng] [TestNG] Running:
   [testng]   C:TestNG_WORKSPACETestNGWithAntsrc    estng.xml
   [testng]
   [testng] Inside testPrintMessage()
   [testng] Manisha
   [testng] Inside testSalutationMessage()
   [testng] Hi!Manisha
   [testng]
   [testng] ===============================================
   [testng] Plug ANT test Suite
   [testng] Total tests run: 2, Failures: 0, Skips: 0
   [testng] ===============================================
   [testng]

BUILD SUCCESSFUL
Total time: 1 second
```

# TestNG Eclipse插件 - TestNG教程

用eclipse设置TestNG，下面的步骤必须遵循：

## 步骤1：下载TestNG的归档文件

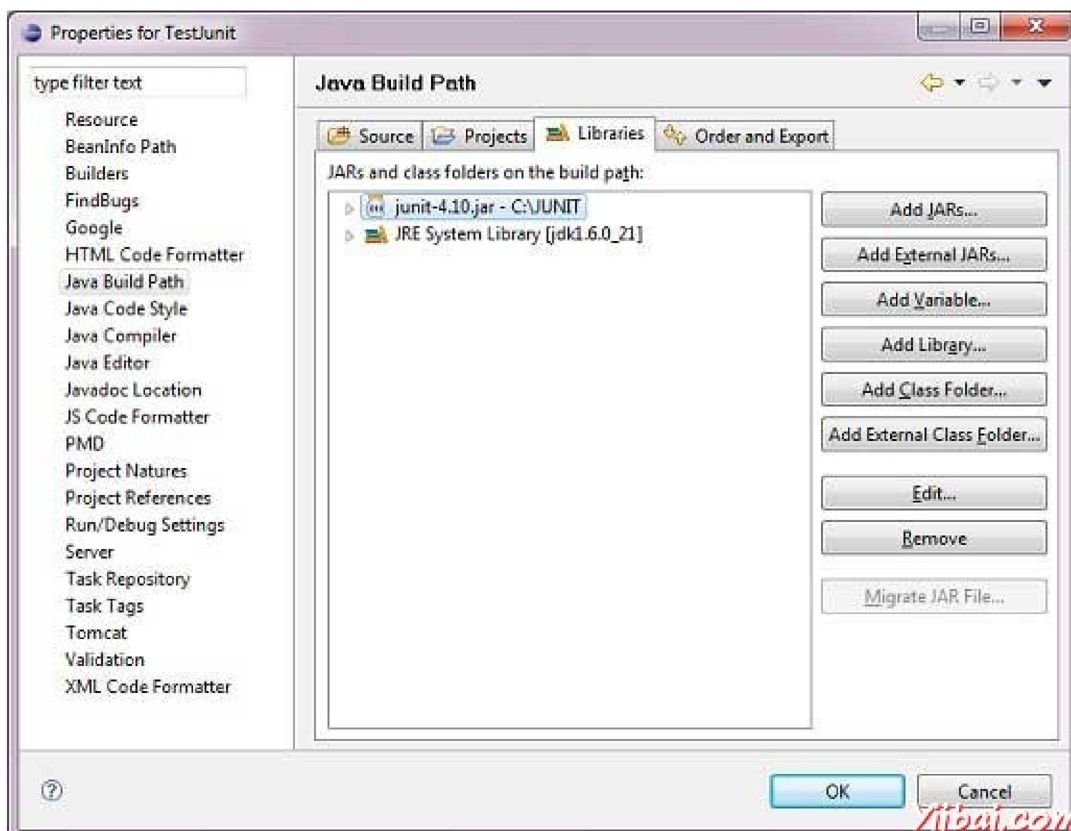
下载 http://www.testng.org

| OS | 压缩文件名 |
|---|---|
| Windows | testng-6.8.jar |
| Linux | testng-6.8.jar |
| Mac | testng-6.8.jar |

假设你上面复制的JAR文件到 C:>TestNG 文件夹.

## 第二步：设置Eclipse环境

- 打开 eclipse -> 右键单击项目，然后单击property > Build Path > Configure Build Path 并添加 testng-6.8.jar 在库中使用 Add External Jar 按钮.



- 我们假设你的eclipse 中 TestNG插件已经内置，如果不是，那么请使用更新站点获取最新版本：
    - 在你的 eclipse IDE, 选择 Help / Software updates / Find and Install.
    - 搜索新功能安装。
    - 新的远程站点。
    - For Eclipse 3.4 and above, enter http://beust.com/eclipse.
    - For Eclipse 3.3 and below, enter http://beust.com/eclipse1.
    - Make sure the check box next to URL is checked and click Next.
    - 然后Eclipse会引导帮您完成整个过程。

现在，你的eclipse已经可以使用 TestNG测试用例的开发做好准备。

## 步骤3：确认Eclipse已经安装TestNG

- 在eclipse中创建一个项目TestNGProject
- 创建一类MessageUtil在项目测试。

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {

   private String message;

   //Constructor
   //@param message to be printed
   public MessageUtil(String message){
      this.message = message;
   }

   // prints the message
   public String printMessage(){
      System.out.println(message);
      return message;
   }
}
```

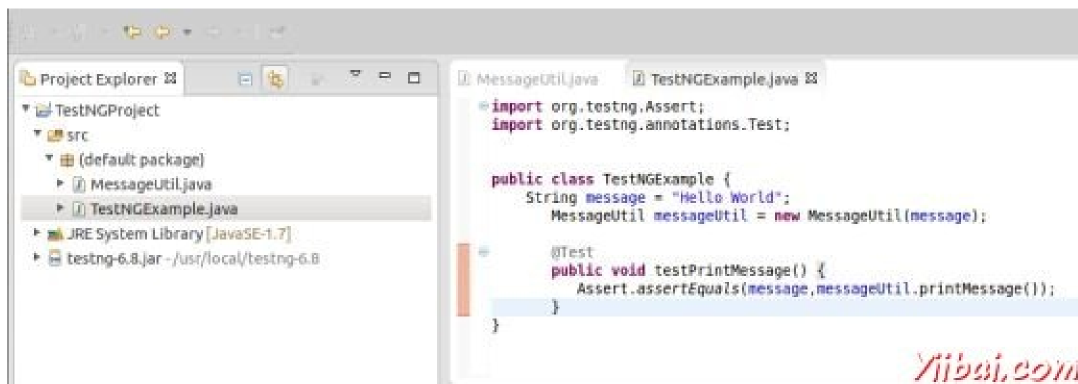
- 在项目中创建一个测试类TestNGExample

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class TestNGExample {
    String message = "Hello World";
    MessageUtil messageUtil = new MessageUtil(message);

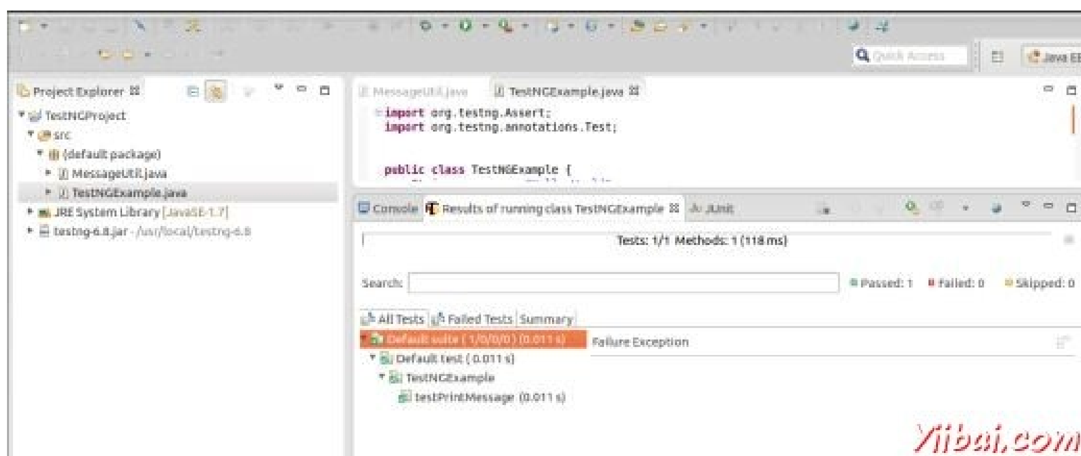
    @Test
    public void testPrintMessage() {
       Assert.assertEquals(message,messageUtil.printMessage());
     }
}
```

下面应该是项目结构：




最后，通过右击程序和TestNG的运行验证程序的输出。

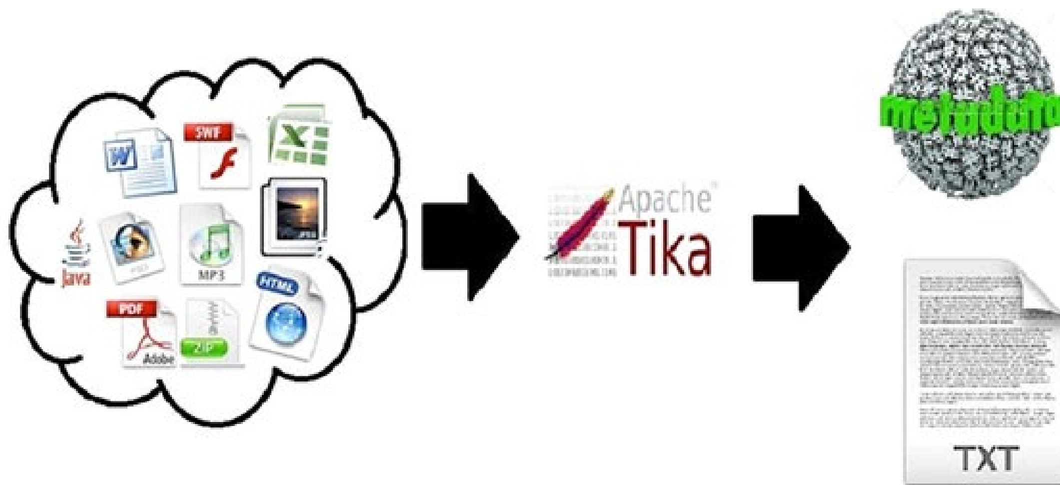
验证结果。



# Tika教程

## Apache Tika 是什么?

- Apache Tika用于文件类型检测和从各种格式的文件内容提取的库。
- 在内部，Tika使用现有的各种文件解析器和文档类型的检测技术来检测和提取数据。
- 使用Tika，人们可以开发出通用型检测器和内容提取到的不同类型的文件，如电子表格，文本文件，图像，PDF文件甚至多媒体输入格式，在一定程度上提取结构化文本以及元数据。
- Tika提供用于解析不同文件格式的一个通用API。它采用83个现有的专业解析器库，为每个文档类型。
- 所有这些解析器库是根据一个叫做Parser接口单一接口封装。




## 为什么用Tika?

据filext.com网站统计，大约有1.5万至51K的内容类型，并且这个数字还在与日俱增。数据被存储在不同的格式，如文本文档，excel表格，PDF，图像和多媒体文件，仅举几例。因此，应用程序如搜索引擎和内容管理系统需要从这些文档类型容易提取数据的额外的支持。Apache Tika 通过提供一个通用的API来检测并提取多种文件格式的数据服务达到这一目的。

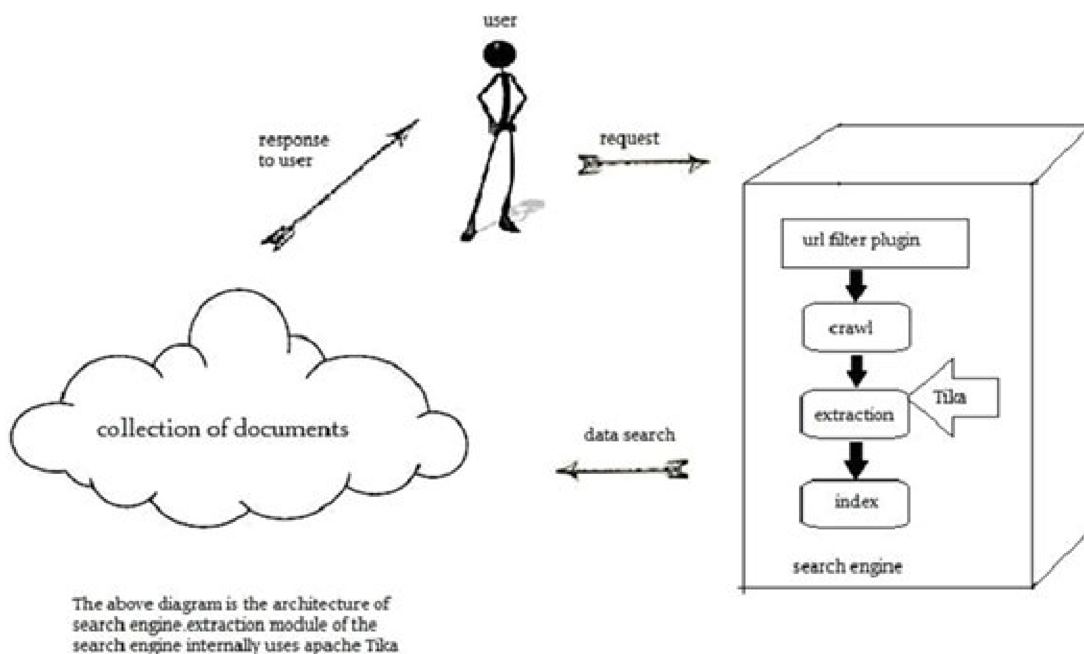
## Apache Tika 应用

有各种各样的应用程序使用Apache Tika。在这里，我们将讨论严重依赖Apache Tika几个突出的应用。

## 搜索引擎

开发搜索引擎索引的数字文档的文本内容使Tika被广泛使用。

- 搜索引擎是用于搜索的网页信息和索引文件的信息处理系统。
- 抓取工具是通过Web抓取获取使用一些索引技术被索引的文件搜索引擎的重要组成部分。此后，抓取工具传送这些索引文件提取成分。
- 提取成分的职责是提取文档中的文本和元数据。这样提取的内容和元数据是对搜索引擎非常有用。该提取组件包含在Tika中。
- 然后将提取的内容被传递到使用它来建立一个搜索索引搜索引擎的索引器。此外，该搜索引擎使用许多其它方式提取的内容也是如此。




## 文档分析

- 在人工智能领域，有一定的工具来自动分析文件在语义层面，并提取各种数据来自他们。
- 在这种应用中，这些文件是基于在文档的所提取的内容的突出方面进行分类。
- 这些工具使用提Tika内容提取分析从纯文本到不同的数字文档文件。

## 数字资产管理

- 有些组织管理他们的数字资产，如使用一种称为数字资产管理（DAM）的特殊应用程序的照片，电子书，绘图，音乐和视频。

- 这样的应用程序采取的文件类型检测器和元数据提取器的帮助下到的各种文件进行分类。

## 内容分析

- 像亚马逊网站建议根据自己的兴趣刚刚发布了他们的网站内容向个人用户。要做到这一点，这些网站遵循机器学习技术，或采取了类似Facebook的社交媒体网站的帮助下，以提取所需的信息，如喜欢和用户的利益。此收集到的信息将在HTML标签或其他格式需要另外的内容类型检测和提取的形式。

- 为一个文件，内容分析，我们有实现，如UIMA和Mahout的机器学习技术的技术。这些技术是在聚类和分析中的文件中的数据是有用的。

- Apache Mahout是一个框架，它提供基于Apache Hadoop的ML算法- 一个云计算平台。 Mahout 提供了下面的某个集群和过滤技术的架构。按照这个架构，程序员可以编写自己的ML算法，通过采取各种文本和元数据的组合来产生建议。提供输入这些算法，最近Mahout的版本使用Tika提取二进制内容的文本和元数据。

- Apache UIMA 分析和处理各种编程语言，并产生UIMA注解。在内部，它使用提卡注解者抽取文档中的文本和元数据。

## 历史

| 年份 | 开发 |
|---|---|
| 2006 | Tika的想法是在Lucene项目管理委员会之前设计的。 |
| 2006 | Tika及其在Jackrabbit项目有用的概念进行了讨论。 |
| 2007 | Tika进入Apache孵化器。 |
| 2008 | 版本0.1和0.2发布，Tika从孵化器到Lucene子项目独立。 |
| 2009 | 版本0.3，0.4，和0.5发布。 |
| 2010 | 版本0.6和0.7发布，Tika进入Apache的顶级项目。 |
| 2011 | Tika1.0发布，并Tika的书籍“Tika in Action”也在同一年被发布。 |

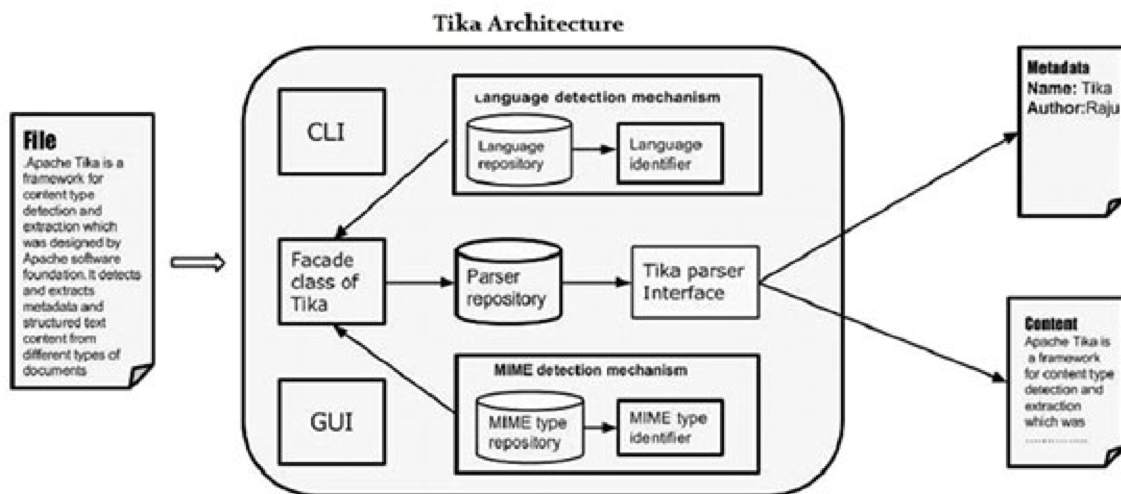
# TIKA架构 - Tika教程

## Tika应用层架构

应用程序员可以很容易地在他们的应用程序集成Tika。Tika提供了一个命令行界面和图形用户界面，使它比较人性化。

在本章中，我们将讨论构成Tika架构的四个重要模块。下图显示了Tika的四个模块的体系结构：

- 语言检测机制。
- MIME检测机制。
- Parser接口。
- Tika Facade 类.




## 语言检测机制

每当一个文本文件被传递到Tika，它将检测在其中的语言。它接受没有语言的注释文件和通过检测该语言添加在该文件的元数据信息。

支持语言识别，Tika 有一类叫做语言标识符在包org.apache.tika.language及语言识别资料库里面包含了语言检测从给定文本的算法。Tika 内部使用N-gram算法语言检测。

## MIME检测机制

Tika可以根据MIME标准检测文档类型。Tika默认MIME类型检测是使用org.apache.tika.mime.mimeTypes。它使用org.apache.tika.detect.Detector 接口大部分内容类型检测。

内部Tika使用多种技术，如文件匹配替换，内容类型提示，魔术字节，字符编码，以及其他一些技术。

## 解析器接口

org.apache.tika.parser 解析器接口是Tika解析文档的主要接口。该接口从提取文档中的文本和元数据，并总结了其对外部用户愿意写解析器插件。

采用不同的具体解析器类，具体为各个文档类型，Tika 支持大量的文件格式。这些格式的具体类不同的文件格式提供支持，无论是通过直接实现逻辑分析器或使用外部解析器库。

### Tika Facade 类

使用的Tika facade类是从Java调用Tika的最简单和直接的方式，而且也沿用了外观的设计模式。可以在 Tika API的org.apache.tika包Tika 找到外观facade类。

通过实现基本用例，Tika作为facade的代理。它抽象了的Tika库的底层复杂性，例如MIME检测机制，解析器接口和语言检测机制，并提供给用户一个简单的接口来使用。

# Tika的特点

- 统一解析器接口：Tika封装在一个单一的解析器接口的第三方解析器库。由于这个特征，用户逸出从选择合适的解析器库的负担，并使用它，根据所遇到的文件类型。
- 低内存占用：Tika因此消耗更少的内存资源也很容易嵌入Java应用程序。也可以用Tika平台像移动那样PDA资源少，运行该应用程序。
- 快速处理：从应用连结内容检测和提取可以预期的。
- 灵活元数据：Tika理解所有这些都用来描述文件的元数据模型。
- 解析器集成：Tika可以使用可在单一应用程序中每个文件类型的各种解析器库。
- MIME类型检测： Tika可以检测并从所有包括在MIME标准的媒体类型中提取内容。
- 语言检测： Tika包括语言识别功能，因此可以在一个多语种网站基于语言类型的文档中使用。

# Tika的功能

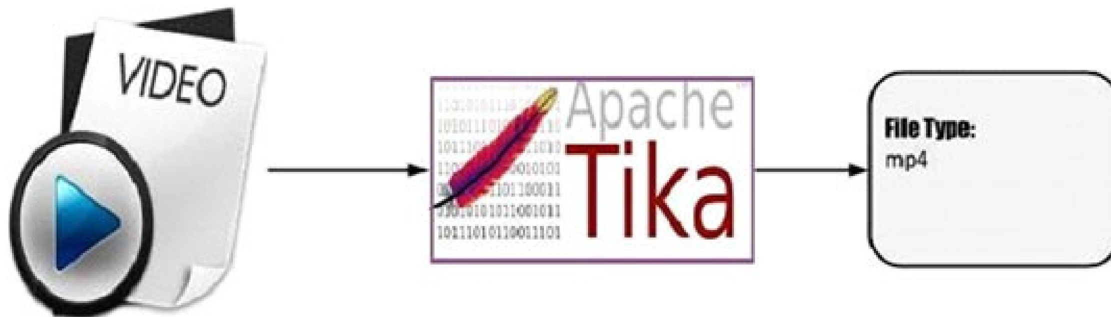
Tika支持多种功能：

- 文档类型检测

- 内容提取
- 元数据提取
- 语言检测

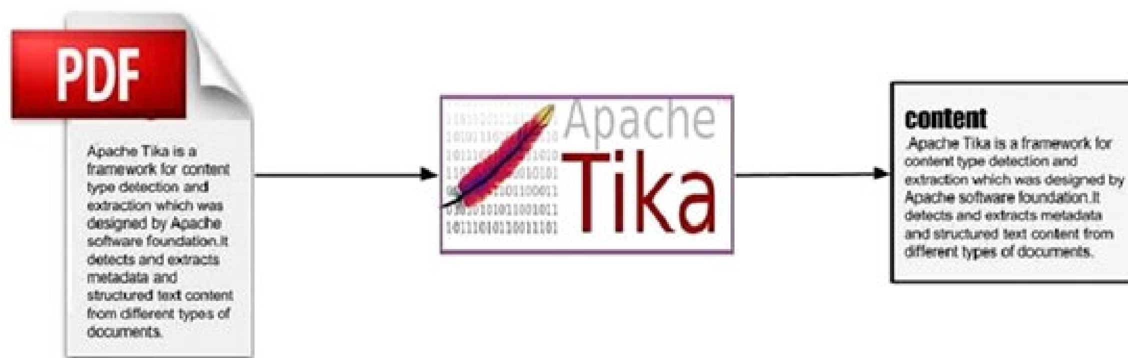
## 文件类型检测

Tika使用不同的检测技术，检测给它的文件的类型。




## 内容提取

Tika有一个解析器库，可以分析各种文档格式的内容，并提取它们。然后检测所述文档的类型，它从解析器库选择的适当的分析器，并传递该文档。不同类别的Tika方法来解析不同的文件格式。



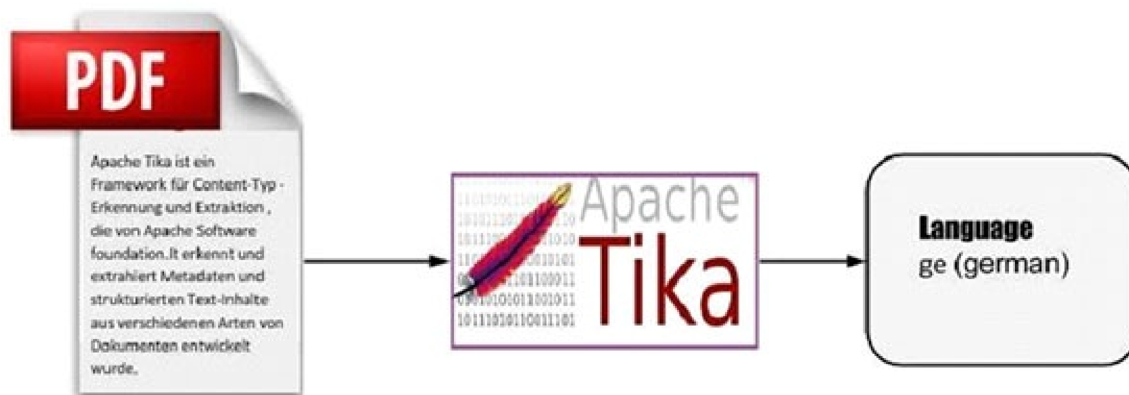

## 元数据提取

随着内容，Tika提取具有相同的程序的文件的元数据中的内容的提取。对于某些文件类型，Tika有接口类提取元数据。



## 语言检测

在内部，Tika如下像一个n-gram算法来检测所述内容的语言的给定文档中。Tika取决于类，如语言识别和Profiler的语言识别。



# TIKA环境配置 - Tika教程

本章将指导完成设置Apache Tika在Windows和Linux的配置过程。用户管理是必要的，同时安装了Apache Tika。

## 系统要求

| JDK | Java SE 2 JDK 1.6 或以上 |
|---|---|
| 内存 | 1 GB RAM (推荐) |
| 硬盘空间 | 无最小要求 |
| 操作系统版本 | Windows XP 或以上, Linux |

### 第1步：验证安装Java

为了验证Java安装，打开控制台并执行下面的Java命令。

| OS | 任务 | 命令 |
|---|---|---|
| Windows | 打开命令控制台 | \>java –version |
| Linux | 打开命令终端 | $java –version |

如果Java已经正确地在您的系统已经安装，那么应该得到以下输出之一，具体取决于您所使用的平台上。

| OS | 输出 |
|---|---|
| Windows | Java version "1.7.0_60"Java (TM) SE Run Time Environment (build 1.7.0_60-b19)Java Hotspot (TM) 64-bit Server VM (build 24.60-b09, mixed mode) |
| Lunix | java version "1.7.0_25"Open JDK Runtime Environment (rhel-2.3.10.4.el6_4-x86_64)Open JDK 64-Bit Server VM (build 23.7-b01, mixed mode) |

- 假设本教程的读者都在继续本教程之前把Java1.7.0_60安装在他们的系统中。
- 如果没有安装Java SDK，从下载其最新版本 http://www.oracle.com/technetwork/java/javase/downloads/index.htmll and have it installed.

### 第2步：设置Java环境

设置JAVA_HOME环境变量指向到安装在机器上的Java基本目录的位置。例如，

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60 |
| Linux | export JAVA_HOME=/usr/local/java-current |

附加Java编译器的位置到系统路径的完整路径。

| OS | 输出 |
|---|---|
| Windows | Append the String; C:\Program Files\Java\jdk1.7.0_60\bin to the end of the system variable PATH. |
| Linux | export PATH=$PATH:$JAVA_HOME/bin/ |

验证命令java版本，命令提示符如上所述。

## 第3步：设置Apache Tika环境

可以在自己的环境中通过使用集成Apache Tika ：

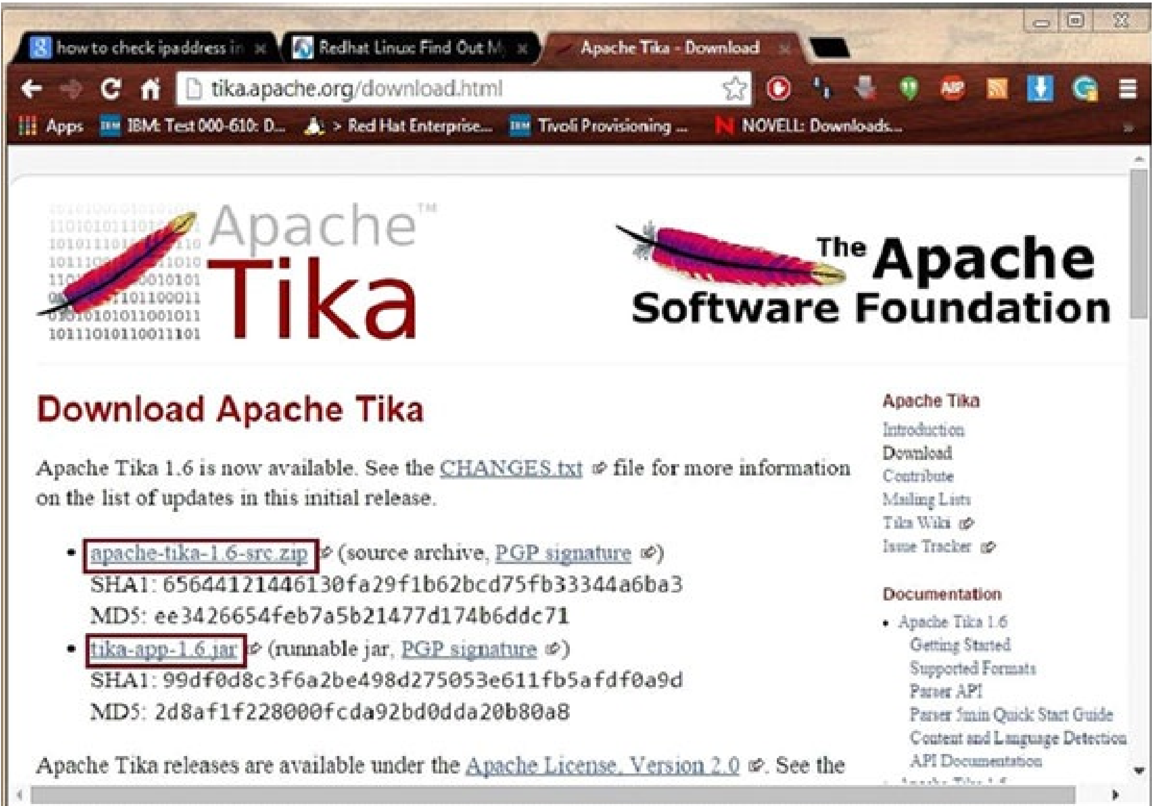
- 命令行，
- Tika API,
- Tika（CLI）的命令行界面，
- Tika的图形用户界面（GUI），或
- 源代码

对于任何一种方法，首先，必须下载的Tika源代码。

Tika的源代码在 http://Tika.apache.org/download.htmll，在那里找到两个链接：

apache-tika-1.6-src.zip: 它包含的Tika的源代码以及 Tika -app-1.6.jar: 它是一个包含Tika应用程序的JAR文件。

下载这两个文件。Tika的官方网站的快照如下所示。




下载文件后，设置类路径的JAR文件 tika-app-1.6.jar。添加 jar 文件的完整路径，如图表所示。

| OS | Output |
| --- | --- |
| Windows | 添加字符串 “C:\jars\Tika-app-1.6.jar” 到用户环境变量 CLASSPATH |
| Linux | Export CLASSPATH=$CLASSPATH:/usr/share/jars/Tika-app-1.6.tar: |

Apache提供Tika应用程序，使用Eclipse的图形用户界面（GUI）应用程序。

# 使用Eclipse构建Tika-Maven

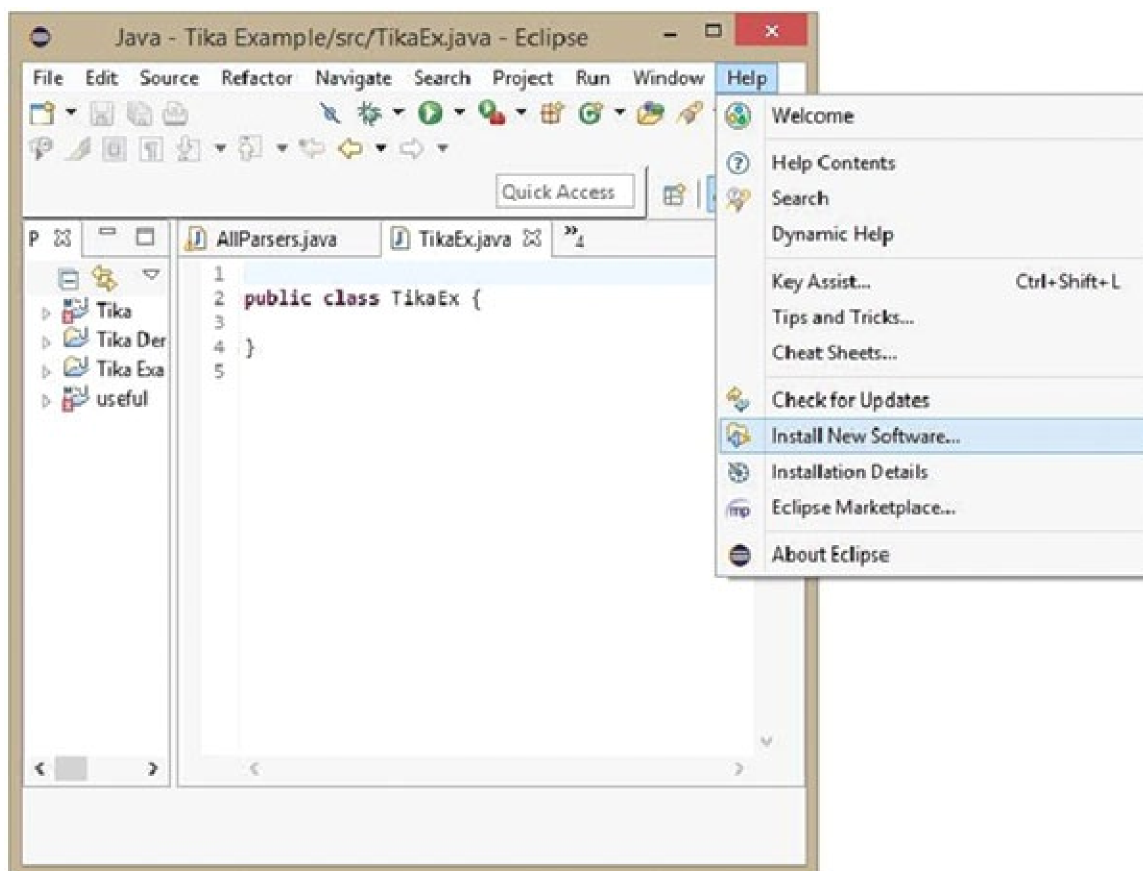
- 打开Eclipse，并创建一个新的项目。
- 如果在Eclipse中没有Maven，按照给定的步骤进行设置。
    - 打开 link http://wiki.eclipse.org/M2E_updatesite_and_gittags. 在那里，会发现m2e插件版本以表格格式

m2e releases

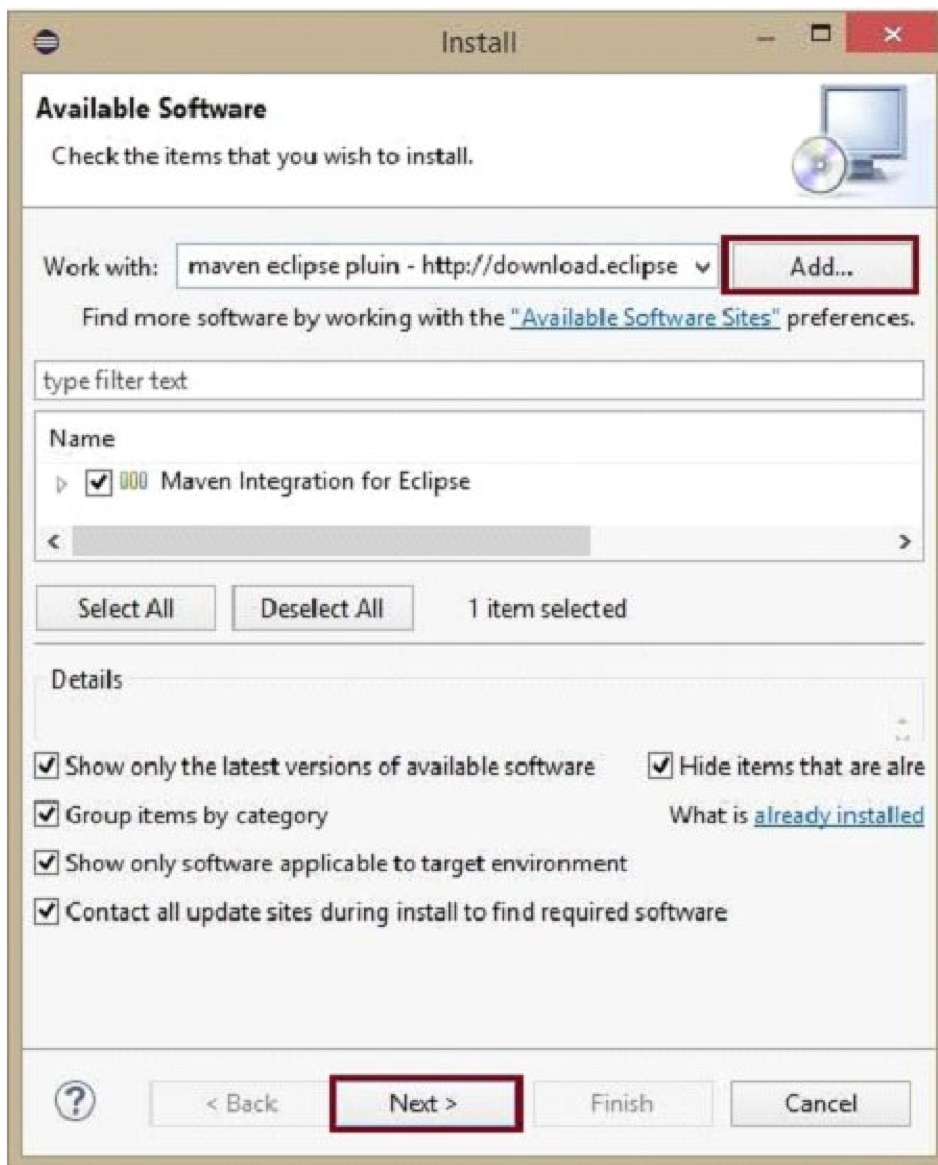
| release | full version | date | tag | p2 url |
|---|---|---|---|---|
| 1.0 | 1.0.0.20110607-2117 | 2011-06-22 | releases/1.0/1.0.0.20110607-2117 | http://download.eclipse.org/technology/m2e/releases/1.0/1.0.0.20110607-2117 |
| 1.0 SR1 | 1.0.100.20110804-1717 | 2011-09-23 | releases/1.0/1.0.100.20110804-1717 | http://download.eclipse.org/technology/m2e/releases/1.0/1.0.100.20110804-1717 |
| 1.0 SR2 | 1.0.200.20111228-1245 | 2012-02-24 | releases/1.0/1.0.200.20111228-1245 | http://download.eclipse.org/technology/m2e/releases/1.0/1.0.200.20111228-1245 |
| 1.1 | 1.1.0.20120530-0009 | 2012-06-27 | releases/1.1/1.1.0.20120530-0009 | http://download.eclipse.org/technology/m2e/releases/1.1/1.1.0.20120530-0009 |
| 1.2 | 1.2.0.20120903-1050 | 2012-09-19 | releases/1.2/1.2.0.20120903-1050 | http://download.eclipse.org/technology/m2e/releases/1.2/1.2.0.20120903-1050 |
| 1.3 | 1.3.0.20130129-0926 | 2013-02-20 | releases/1.3/1.3.0.20130129-0926 | http://download.eclipse.org/technology/m2e/releases/1.3/1.3.0.20130129-0926 |
| 1.3.1 | 1.3.1.20130219-1424 | 2013-02-20 | releases/1.3/1.3.1.20130219-1424 | http://download.eclipse.org/technology/m2e/releases/1.3/1.3.1.20130219-1424 |
| 1.4 | 1.4.0.20130601-0317 | 2013-06-26 | releases/1.4/1.4.0.20130601-0317 | http://download.eclipse.org/technology/m2e/releases/1.4/1.4.0.20130601-0317 |
| 1.4.1 | 1.4.1.20140328-1905 | 2014-03-31 | releases/1.4/1.4.1.20140328-1905 | http://download.eclipse.org/technology/m2e/releases/1.4/1.4.1.20140328-1905 |
| 1.5 | 1.5.0.20140606-0033 | 2014-06-25 | releases/1.5/1.5.0.20140606-0033 | http://download.eclipse.org/technology/m2e/releases/1.5/1.5.0.20140606-0033 |

Retrieved from "http://wiki.eclipse.org/index.php?title=M2E_updatesite_and_gittags&oldid=365954"
Category:

- 挑选的最新版本，并保存在URL网址的p2列的路径。
- 现在重新访问eclipse，在菜单栏中，单击帮助，然后从下拉菜单中选择安装新软件




- 单击 Add 按钮，输入任何想要的名称，因为它是可选的。现在贴在位置字段中保存的 URL。
- 一个新的插件选择在上一步中添加的名字，选中复选框在它前面，然后单击下一步 Next。




- 继续安装。完成后，重新启动Eclipse。
- 现在，右键单击该项目，并在配置选项，选择Convert to Maven项目。
- 将出现创建一个新的POM新的向导。输入组ID为org.apache.tika，进入Tika的最新版本，选择jar 包，然后单击Finish。

Maven项目已成功安装，并且项目转化成Maven。现在，必须配置pom.xml文件。

## 配置XML文件

可以通过Tika Maven的依赖 http://mvnrepository.com/artifact/org.apache.tika

下面显示的是Apache Tika完整的Maven依赖。

```
<dependency>
   <groupId>org.apache.Tika</groupId>
   <artifactId>Tika-core</artifactId>
   <version>1.6</version>

   <groupId>org.apache.Tika</groupId>
   <artifactId> Tika-parsers</artifactId>
   <version> 1.6</version>

   <groupId> org.apache.Tika</groupId>
   <artifactId>Tika</artifactId>
   <version>1.6</version>

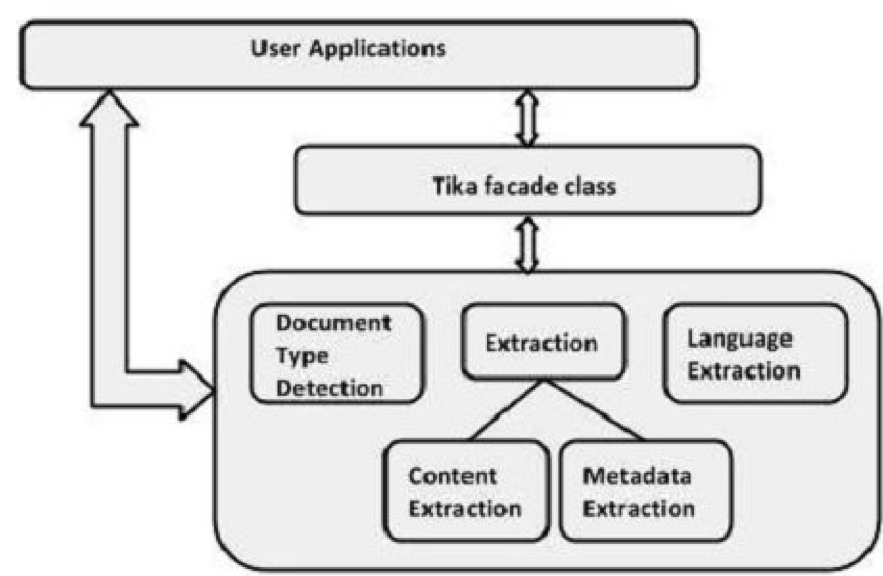
   <groupId>org.apache.Tika</groupId>
   < artifactId>Tika-serialization</artifactId>
   < version>1.6< /version>

   < groupId>org.apache.Tika< /groupId>
   < artifactId>Tika-app< /artifactId>
   < version>1.6< /version>

   <groupId>org.apache.Tika</groupId>
   <artifactId>Tika-bundle</artifactId>
   <version>1.6</version>
</dependency>
```

# TIKA参考API - Tika教程

用户可以在使用 Tika 的外观类在应用程序中嵌入Tika。它的方法来探索Tika的所有功能。因为它是一个外网类，Tika抽象的背后有其功能的复杂性。除了这一点，用户还可以使用各种Tika类在他们的应用程序。




## Tika 类 (facade)

这是最突出的Tika类库和正面设计模式。因此，抽象所有的内部实现，并提供了简单的方法来访问Tika功能。下表列出该类的构造函数以及它们的描述。

package : org.apache.tika

class: Tika

| S.No. | 构造函数和说明 |
|---|---|
| 1 | **Tika ()**使用默认配置，构建Tika类。 |
| 2 | **Tika (Detector detector)**通过接受检测实例作为参数创建 Tika 的外观 |
| 3 | **Tika (Detector detector, Parser parser)**创建一个Tika外观通过接受检测和解析器实例作为参数。 |
| 4 | **Tika (Detector detector, Parser parser, Translator translator)**创建一个Tika外观通过接受检测器，解析器，并且转换实例作为参数。 |
| 5 | **Tika (TikaConfig config)**创建一个Tika外观通过接受TikaConfig类作为参数的对象。 |

## 方法和说明

以下是 Tika外观类的重要方法：

| S.No. | 方法和描述 |
|---|---|
| 1 | **String parseToString (File file)**此方法及其所有变种分析作为参数传递的文件，并返回字符串格式提取的文本内容。默认情况下，这个字符串参数的长度是有限的。 |
| 2 | **int getMaxStringLength ()**返回由parseToString方法返回字符串的最大长度。 |
| 3 | **void setMaxStringLength (int maxStringLength)**设置由parseToString方法返回的字符串的最大长度。 |
| 4 | **Reader parse (File file)**该方法及其所有变型解析作为参数传递的文件，并返回 java.io.Reader对象的形式，所提取的文本内容。 |
| 5 | **String detect (InputStream stream, Metadata metadata)**该方法及其所有变接受InputStream对象和元数据对象作为参数，检测出给定文档的类型，并返回该文件类型的名称作为字符串对象。这种方法提炼使用 Tika 的检测手段。 |
| 6 | **String translate (InputStream text, String targetLanguage)**此方法及其所有变种接受InputStream对象，并表示，我们希望我们的文字被翻译语言的字符串，并把特定的文本所需的语言，尝试自动检测源语言。 |

## 解析器接口

这是Tika包的所有解析器类实现的接口。

package : org.apache.tika.parser

Interface : Parser

## 方法和说明

以下是TikaParser接口的重要方法：

| S.No. | 方法及描述 |
|---|---|
| 1 | **parse (InputStream stream, ContentHandler handler, Metadata metadata, ParseContext context)**这个方法将给定的文档解析到XHTML和SAX事件序列。解析后，将放置在ContentHandler类的对象，并在元数据的类的对象的元数据，所提取的文件的内容。 |

# Metadata 类

这个类实现了各种接口，如素材，地理，HttpHeaders，消息，微软Office，气候预测，TIFF，TikaMetadataKeys，TikaMimeKeys，Serializable接口，支持各种数据模型。下表列出了构造函数和这个类的方法及其说明。

**package :** org.apache.tika.metadata

**class :** Metadata

| S.No. | 构造方法及描述 |
|---|---|
| 1 | **Metadata()**构造一个新的，空的元数据。 |

| S.No. | 方法及描述 |
|---|---|
| 1 | **add (Property property, String value)**增加了一个元数据属性/值映射到给定的文件。使用此功能，可以将该值设置为一个属性。 |
| 2 | **add (String name, String value)**增加了一个元数据属性/值映射到给定的文件。使用这种方法，我们可以使用新名称的值设置为一个文件，从现有的元数据。 |
| 3 | **String get (Property property)**返回给定的元数据属性的值（如果有的话）。 |
| 4 | **String get (String name)**返回给定元数据的名称的值（如果有的话）。 |
| 5 | **Date getDate (Property property)**返回日期的元数据属性的值。 |
| 6 | **String[] getValues (Property property)**返回的元数据属性的所有的值。 |
| 7 | **String[] getValues (String name)**返回给定元数据的名称的所有的值。 |
| 8 | **String[] names()**返回元数据对象的元数据元素的所有的名字。 |
| 9 | **set (Property property, Date date)**设置给定的元数据属性的日期值 |
| 10 | **set(Property property, String[] values)**设置多个值到一个元数据属性。 |

# 语言类标识符

此分类标识了特定内容的语言。下表列出了这个类的构造函数以及它们的描述。

**package :** org.apache.tika.language

**class :** Language Identifier

| S.No. | 构造器和说明 |
|---|---|
| 1 | **LanguageIdentifier (LanguageProfile profile)**实例化的语言标识符。在这里必须通过一个LanguageProfile对象作为参数。 |
| 2 | **LanguageIdentifier (String content)**这个构造函数可以通过从文本内容传递一个String实例化一个语言标识符 |

| S.No. | 构造器和说明 |
|---|---|
| 1 | **String getLanguage ()**返回给当前LanguageIdentifier对象的语言。 |

# TIKA文件格式 - Tika教程

## Tika支持的文件格式

下面的表显示了Tika支持的文件格式。

| 文件格式 | 类库 |
|---|---|
| XML | org.apache.tika.parser.xml |
| HTML | org.apache.tika.parser.htmll and it uses Tagsoup Library |
| MS-Office compound document Ole2 till 2007 ooxml 2007 onwards | org.apache.tika.parser.microsoftorg.apache.tika.parser.micros and it uses Apache Poi library |
| OpenDocument Format openoffice | org.apache.tika.parser.odf |
| portable Document Format(PDF) | org.apache.tika.parser.pdf and this package uses Apache Pdf library |
| Electronic Publication Format (digital books) | org.apache.tika.parser.epub |
| Rich Text format | org.apache.tika.parser.rtf |
| Compression and packaging formats | org.apache.tika.parser.pkg and this package uses Common c library |
| Text format | org.apache.tika.parser.txt |
| Feed and syndication formats | org.apache.tika.parser.feed |
| Audio formats | org.apache.tika.parser.audio and org.apache.tika.parser.mp3 |
| Imageparsers | org.apache.tika.parser.jpeg |
| | |

| | |
|---|---|
| Videoformats | org.apache.tika.parser.mp4 and org.apache.tika.parser.video parser internally uses Simple Algorithm to parse flash video fo |
| java class files and jar files | org.apache.tika.parser.asm |
| Mobxformat (email messages) | org.apache.tika.parser.mbox |
| Cad formats | org.apache.tika.parser.dwg |
| FontFormats | org.apache.tika.parser.font |
| executable programs and libraries | org.apache.tika.parser.executable |

# TIKA文件类型检测 - Tika教程

## MIME 标准

多用途Internet邮件扩展(MIME)标准，用于识别文件类型的最佳标准。这些标准的知识有助于在内部相互作用的浏览器。

当浏览器遇到一个媒体文件，它选择可用它来显示其内容的兼容软件。在情况下，它不具有任何合适的应用程序，以运行一个特定媒体文件，它建议用户获得合适的插件软件。

## Tika类型检测

Tika支持MIME所提供的所有互联网媒体文件类型。每当一个文件通过Tika检测到该文件，其文件类型。检测的介质类型，Tika内部通过以下机制。

### 文件扩展名

检查文件扩展名是检测的文件的格式的最简单和最广泛使用的方法。许多应用程序和操作系统提供这些扩展的支持。下面所示是一些已知文件类型的扩展名。

| 文件名 | 扩展名 |
|---|---|
| image | .jpg |
| audio | .mp3 |
| java archive file | .jar |
| java class file | .class |

### 内容类型提示

每当从数据库中检索文件或将其附加到另一个文档，可能会失去该文件的名称或扩展名。在这种情况下，该文件所提供的元数据被用于检测文件的扩展名。

### 魔术字节

遵守文件的原始字节，可以为每个文件找到一些独特的字符模式。一些文件具有特殊的字节前缀称为被专门制成并包含在一个文件中，用于识别文件类型的目的魔术字节。

例如，可以找到CA FE BA在一个PDF文件（十六进制格式）的一个java文件和PDF %（ASCII格式）。 Tika使用此信息来识别的文件的媒体类型。

## 字符编码

文件纯文本使用不同类型的字符编码的编码。这里的主要挑战是确定在文件中使用的字符编码的类型。Tika 一样的标记和字节的频率来识别所使用的纯文本内容的编码系统字符编码技术。

## XML根字符

为了检测XML文档，Tika解析XML文档并提取，如根元素，命名空间和引用的架构，从文件的真实介质类型，可以找到的信息。

# 使用Facade类类型检测

facade类的detect() 方法被用于检测文档类型。这个方法接受一个文件作为输入。下面显示的是文件类型检测与Tika外观类的示例程序。

```
import java.io.File;

import org.apache.tika.Tika;

public class Typedetection {

   public static void main(String[] args) throws Exception {

      //assume example.mp3 is in your current directory
      File file = new File("example.mp3");//

      //Instantiating tika facade class
      Tika tika = new Tika();

      //detecting the file type using detect method
      String filetype = tika.detect(file);
      System.out.println(filetype);
   }
}
```

将以上代码保存为TypeDetection.java并在命令提示符上使用以下命令运行它：

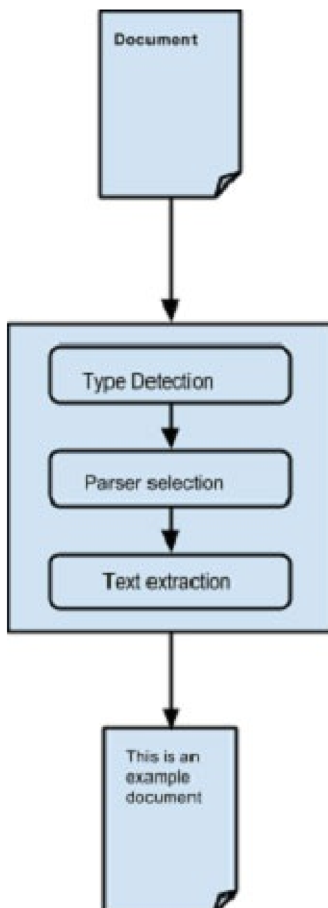
```
javac TypeDetection.java
java TypeDetection

audio/mpeg
```

# TIKA内容提取 - Tika教程

Tika使用不同的解析器库来提取给解析器的内容。它选择了正确的语法分析器提取给定的文档类型。

解析文件，一般用于Tika外观facade类的parseToString()方法。下面显示的是所涉及分析过程的步骤和这些由Tika的ParsertoString()方法提取。




抽象的分析过程：

- 最初，当我们传一个文件到Tika，它使用与之适合的类型检测机制和检测文件类型。
- 一旦文档类型是已知的，它选择从解析器库中合适的解析器。解析器库中包含的类使用外部库。
- 然后将文档传送到选择将解析的内容，提取文本，并且还抛出了不可读格式异常解析器。

# 使用Tika内容提取

下面给出的是程序使用Tika facade 类从文件中提取文本：

```
import java.io.File;
import java.io.IOException;

import org.apache.tika.Tika;
import org.apache.tika.exception.TikaException;

import org.xml.sax.SAXException;

public class TikaExtraction {

   public static void main(final String[] args) throws IOException,

      //Assume sample.txt is in your current directory
      File file = new File("sample.txt");

      //Instantiating Tika facade class
      Tika tika = new Tika();
      String filecontent = tika.parseToString(file);
      System.out.println("Extracted Content: " + filecontent);
   }
}
```

将以上代码保存为TikaExtraction.java并在命令提示符下运行：

```
javac TikaExtraction.java
java TikaExtraction
```

注意：假设 sample.txt 是具有下列内容。

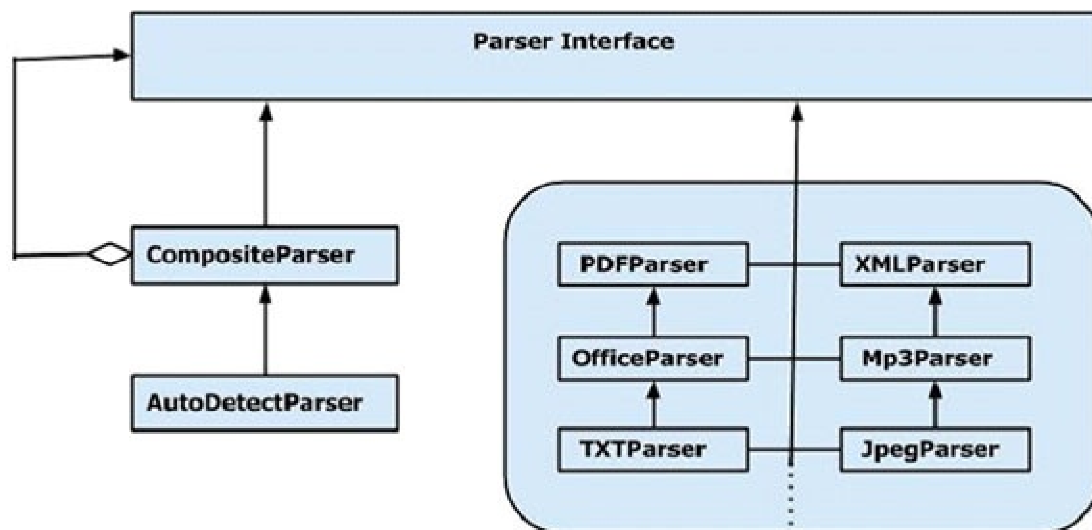
```
Hi students welcome to yiibai
```

它提供了以下的输出：

```
Extracted Content: Hi students welcome to yiibai
```

# 使用Parser接口内容提取

Tika 解析器包提供了使用它可以分析一个文本文档的几个接口和类。下面给出的是org.apache.tika.parser包的框架图。




有几个可用的解析器类，如PDF格式分析器，Mp3Passer，OfficeParser等，逐一分析各自的文件。所有这些类都实现了解析器接口。

## CompositeParser

给出的图表显示Tika通用解析器类CompositeParser 主AutoDetectParser。由于CompositeParser类遵循复合设计模式，可以用一组解析器实例作为一个单独的解析器。CompositeParser类也可以访问所有实现解析器接口的类。

## AutoDetectParser

这是CompositeParser的子类，它提供了自动类型检测。使用此功能，AutoDetectParser自动发送收到的文件到使用该复合方法适当分析器类。

## parse()方法

除了parseToString()，还可以使用分析器接口的parse()方法。该方法的原型如下所示。

```
parse([InputStream](http://docs.oracle.com/javase/6/docs/api/java/i
[ParseContext](https://tika.apache.org/1.6/api/org/apache/tika/pars
```

下表列出了它接受作为参数的四个对象。

| S.No. | 对象及描述 |
|---|---|
| 1 | **InputStream stream**包含任何文件的InputStream对象的内容 |
| 2 | **ContentHandler handler**Tika通过文档作为XHTML内容到此处理，此后该文件正在使用SAX API处理。它提供了在一个文件有效的后处理的内容。 |
| 3 | **Metadata metadata**元数据对象是用来既作为源和文件的元数据的目标。 |
| 4 | **ParseContext context**此对象使用在如遇有客户端应用程序想要定制解析过程。 |

## 例如：

下面给出一个例子，说明如何使用 parse()方法。

步骤 **1**：

要使用解析器接口的parse()方法，实例化任何为其提供实现这个接口的类。

也有个别解析器类，如PDFParser，OfficeParser，XMLParser等等。可以使用这些个人文件解析器。或者也可以使用CompositeParser或AutoDetectParser在内部使用的所有解析器类，并提取使用合适的解析器文档的内容。

```
Parser parser = new AutoDetectParser();
   (or)
Parser parser = new CompositeParser();
   (or)
object of any individual parsers given in Tika Library
```

步骤 **2**：

创建一个处理类的对象。下面给出的是三个内容处理程序：

| S.No. | 类及描述 |
|---|---|
| 1 | **BodyContentHandler**这个类采用XHTML输出的主体部分，并写入该内容到输出写入或输出流。然后重定向XHTML内容到另一个内容处理程序实例。 |
| 2 | **LinkContentHandler**这个类检测，并挑选XHTML文档的所有H-参考标签和转发那些使用类似网络爬虫工具。 |
| 3 | **TeeContentHandler**这个类可以帮助在同时使用多个工具。 |

由于我们的目标是要提取的文件的文本内容，实例化BodyContentHandler如下图所示：

```
BodyContentHandler handler = new BodyContentHandler( );
```

步骤 **3**：

创建的元数据对象，如下所示：

```
Metadata metadata = new Metadata();
```

步骤 **4**：

创建任何输入流对象，并通过您的文件应该被提取到它。

## FileInputstream

通过将文件路径作为参数实例化一个文件对象，这个对象传递给的FileInputStream类的构造函数。

注意：传递给文件对象的路径不应包含空格。

使用这些输入流类的问题是，它们不支持随机访问读取，来高效地处理某些文件格式是必需。要解决此问题，Tika提供TikaInputStream。

```
File file=new File(filepath)
FileInputStream inputstream=new FileInputStream(file);
   (or)
InputStream stream = TikaInputStream.get(new File(filename));
```

步骤 **5**：

创建一个解析的上下文对象，如下所示：

```
ParseContext context =new ParseContext();
```

步骤 **6**：

实例化解析器对象，调用parse方法，并通过所有需要的对象，如下面的原型：

```
parser.parse(inputstream, handler, metadata, context);
```

下面给出的是程序使用的解析器接口内容提取：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class ParserExtraction {

public static void main(final String[] args) throws IOException,SA

   //Assume sample.txt is in your current directory
   File file = new File("sample.txt");

   //parse method parameters
   Parser parser = new AutoDetectParser();
   BodyContentHandler handler = new BodyContentHandler();
   Metadata metadata = new Metadata();
   FileInputStream inputstream = new FileInputStream(file);
   ParseContext context = new ParseContext();

   //parsing the file
   parser.parse(inputstream, handler, metadata, context);
   System.out.println("File content : " + Handler.toString());
   }
}
```

将以上代码保存为 ParserExtraction.java 并在命令提示符下运行：

```
javac  ParserExtraction.java
java  ParserExtraction
```

假设 sample.txt 包含以下内容。

```
Hi students welcome to yiibai
```

它提供了以下的输出：

```
File content : Hi students welcome to yiibai
```

# TIKA元数据提取 - Tika教程

除了内容，Tika还可以从一个文件中提取元数据。元数据是什么，但用文件所提供的附加信息。如果我们考虑一个音频文件，艺术家名，专辑名，标题下自带的元数据。

## XMP标准

可扩展元数据平台(XMP)是用于处理和涉及到的文件的内容存储信息的标准。它是由Adobe系统公司的XMP创建提供了用于定义，创建和元数据的处理标准。可以嵌入该标准为多种文件格式，如PDF，JPEG，JPEG，GIF，JPG，HTML等。

## Property 类

Tika使用属性类遵循XMP属性定义。它提供了PropertyType和值类型枚举捕获的元数据的名称和值。

## Metadata 类

这个类实现了各种接口，如ClimateForcast, CativeCommons,Geographic, TIFF 等提供各种元数据模型的支持。此外，此类提供各种方法来提取一个文件的内容。

## Metadata 名称

我们可以从它的元数据对象用的方法()提取一个文件的所有元数据的名称的列表。它返回所有的名字作为一个字符串数组。使用元数据的名称，就可以得到使用get()方法的值。它需要一个元数据的名称，并返回与它相关联的值。

```
String[] metadaNames = metadata.names();

String value = metadata.get(name);
```

## 使用解析法提取元数据

当我们分析一个使用文件parse()，传递一个空的元数据对象作为一个参数。这种方法提取指定的文件的元数据(如果该文件中包含有)，并将它们放置在元数据对象。因此，在使用parse()解析文件后，就可以提取该对象的元数据。

```
Parser parser = new AutoDetectParser();
BodyContentHandler handler = new BodyContentHandler();
Metadata metadata = new Metadata();   //empty metadata object
FileInputStream inputstream = new FileInputStream(file);
ParseContext context = new ParseContext();
parser.parse(inputstream, handler, metadata, context);
// now this metadata object contains the extracted metadata of the
metadata.metadata.names();
```

下面给出的是一个完整的程序，从文本文件中提取元数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class GetMetadata {

   public static void main(final String[] args) throws IOException,

      //Assume that boy.jpg is in your current directory
      File file=new File("boy.jpg");

      //Parser method parameters
      Parser parser = new AutoDetectParser();
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(file);
      ParseContext context = new ParseContext();

      parser.parse(inputstream, handler, metadata, context);
      System.out.println(handler.toString());

      //getting the list of all meta data elements
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ": " + metadata.get(name));
      }
   }
}
```

将以上代码保存为GetMetadata.java并在命令提示符处使用以下命令运行它：

```
javac  GetMetadata .java
java  GetMetadata
```

注意：假设下面的图片是boy.jpg




它提供了以下的输出：

```
Resolution Units: inch
Compression Type: Baseline
Data Precision: 8 bits
Number of Components: 3
tiff:ImageLength: 1000
Component 2: Cb component: Quantization table 1, Sampling factors 1
Component 1: Y component: Quantization table 0, Sampling factors 1
Image Height: 1000 pixels
X Resolution: 300 dots
Original Transmission Reference: 53616c7465645f5fd22a84941585d89cc7
Image Width: 714 pixels
IPTC-NAA record: 92 bytes binary data
Component 3: Cr component: Quantization table 1, Sampling factors 1
tiff:BitsPerSample: 8
Application Record Version: 4
tiff:ImageWidth: 714
Content-Type: image/jpeg
Y Resolution: 300 dots
```

还可以得到想要的元数据值。

## 添加新的元数据值

可以添加使用元数据类的add()方法新的元数据值。下面给出的是该方法的语法。在这里要添加的作者名字。

```
metadata.add(“author”,”Tutorials yiibai”);
```

元数据Metadata类预定义的属性包括类，如ClimateForcast，CativeCommons，Geographic继承等，以支持各种数据模型的属性。下面示出的是从由提卡实施遵循XMP元数据标准的TIFF图像格式的TIFF接口继承了软件数据类型的使用。

```
metadata.add(Metadata.SOFTWARE,"ms paint");
```

下面给出的是演示如何将元数据值添加到一个给定文件的完整程序。这里的元数据元素的列表被显示在输出，这样就可以增加新的值之后，观察在列表中的变化。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Arrays;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class AddMetadata {

   public static void main(final String[] args) throws IOException,

      //create a file object and assume sample.txt is in your curre
      File file = new File("Example.txt");

      //Parser method parameters
      Parser parser = new AutoDetectParser();
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(file);
      ParseContext context = new ParseContext();

      //parsing the document
      parser.parse(inputstream, handler, metadata, context);

      //list of meta data elements before adding new elements
      System.out.println( " metadata elements :"  +Arrays.toString(

      //adding new meta data name value pair
      metadata.add("Author","Tutorials Point");
      System.out.println(" metadata name value pair is successfully

      //printing all the meta data elements after adding new elemen
      System.out.println("Here is the list of all the metadata elem
      System.out.println( Arrays.toString(metadata.names()));
   }
}
```

将以上代码保存为AddMetadata.java类，然后从命令提示符下运行它：

```
javac  AddMetadata .java
java  AddMetadata
```

假设 example.txt 有下列内容：

```
Hi students welcome to yiibai
```

它提供了以下的输出：

```
metadata elements of the given file :[Content-Encoding, Content-Typ

metadata name value pair is successfully added
Here is the list of all the metadata elements  after adding new ele
[Content-Encoding, Author, Content-Type]
```

## 设定值，用现有的元数据元素

可以设置值，使用set()方法在现有的元数据元素。使用set()方法设置的时间属性的语法如下：

```
metadata.set(Metadata.DATE, new Date());
```

还可以设置多个值，以使用set()方法的属性。使用set()方法多个值设定为作者属性的语法如下：

```
metadata.set(Metadata.AUTHOR, "ram ,raheem ,robin ");
```

下面给出的是一个完整的程序演示set()方法。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import java.util.Date;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class SetMetadata {
```

```
   public static void main(final String[] args) throws IOException,

      //Create a file object and assume example.txt is in your curr
      File file = new File("example.txt");

      //parameters of parse() method
      Parser parser = new AutoDetectParser();
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(file);
      ParseContext context = new ParseContext();

      //Parsing the given file
      parser.parse(inputstream, handler, metadata, context);

      //list of meta data elements elements
      System.out.println( " metadata elements and values of the giv
      String[] metadataNamesb4 = metadata.names();

      for(String name : metadataNamesb4) {
          System.out.println(name + ": " + metadata.get(name));
      }

      //setting date meta data
      metadata.set(Metadata.DATE, new Date());

      //setting multiple values to author property
      metadata.set(Metadata.AUTHOR, "ram ,raheem ,robin ");

      //printing all the meta data elements with new elements
      System.out.println("List of all the metadata elements  after
      String[] metadataNamesafter = metadata.names();

      for(String name : metadataNamesafter) {
         System.out.println(name + ": " + metadata.get(name));
      }
   }
}
```

将以上代码保存为SetMetadata.java并在命令提示符下运行：

```
javac  SetMetadata.java
java  SetMetadata
```

注意：假设sample.txt 有下列内容：

```
Hi students welcome to yiibai
```

在输出中，可以看到新添加的元数据元素。

```
metadata elements and values of the given file :
Content-Encoding: ISO-8859-1
Content-Type: text/plain; charset=ISO-8859-1
Here is the list of all the metadata elements  after adding new ele
date: 2014-09-24T07:01:32Z
Content-Encoding: ISO-8859-1
Author: ram, raheem, robin
Content-Type: text/plain; charset=ISO-8859-1
```

# TIKA语言检测 - Tika教程

## 必要的语言检测

对于基于它们写在一个多语种网站的语言文件分类，语言检测工具是必要的。这个工具应该接受无语言注释(元数据)的文件，并通过检测语言中添加这些信息在文档的元数据。

## 算法性能分析语料库

### 什么是语料库？

为了检测一个文档的语言，语言信息被构造和用已知的语言的信息进行比较。设置这些已知的语言文字被称为语料库。

语料库是一种书面语言，解释了语言用于实际生活中的文本的集合。

语料库是从书本，成绩单，而像其他的互联网数据资源开发。语料库的精度取决于我们使用帧性能分析算法。

### 什么是性能分析算法？

检测语言的常用方法是使用字典。在给定的一段文本中使用的词语将与词典进行匹配。

在语言中常用的单词的列表将是最简单有效的语料库用于检测特定的语言，例如，a, an,在英文中。

### 使用Word设置为语料库

使用字集，一个简单的算法是框架找到两个语料库，其将等于的匹配单词的频率之间的差异的总和之间的距离。

此类算法有以下问题：

- 因为匹配单词的频率非常少，则该算法不能有效地与几个句子小文本工作。它需要大量的文字进行准确匹配。
- 它不能检测到字边界具有语言复合句，而那些具有类似空格或标点符号无字分隔。

由于在使用字集作为语料库这些困难，单个字符或字符组会被考虑。

### 使用字符集为主体

因为这是在一种语言中常用的字符是有限的数目，很容易应用基于单词的频率，而不是字符的算法。这种算法的工作更为出色的情况下在一个或极少数语言中使用的特定的字符集。

此算法存在下列缺点：

- 它是难以区分具有相似性质的频率两种语言。
- 没有任何特定的工具或算法来具体确定的帮助下（如文集）所使用的多语言字符集的语言。

## N-gram算法

上述的缺点就产生了利用给定长度的字符序列为分析语料的一种新方法。个字符的这样的序列被称为N-gram，在一般情况下，N表示该字符序列的长度。

- N元算法是一种有效的方法来检测语言，特别是在案件欧洲语言如英语的。
- 该算法与短文正常工作。
- 虽然有高级语言纹算法来检测多个语言中具有更吸引人的特征的多语言文档，Tika使用3-grams算法，因为它适合于大多数实际情况。

# Tika语言检测

在所有由ISO639-1标准的184标准语言，Tika可检测18种语言。语言检测Tika是通过使用LanguageIdentifier类的getLanguage()方法。此方法返回字符串格式的语言代号。下面给出由Tika检测出的18语言代码对的列表中：

| da—Danish | de—German | et—Estonian | el—Greek |
|---|---|---|---|
| en—English | es—Spanish | fi—Finnish | fr—French |
| hu—Hungarian | is—Icelandic | it—Italian | nl—Dutch |
| no—Norwegian | pl—Polish | pt—Portuguese | ru—Russian |
| sv—Swedish | th—Thai | | |

实例化LanguageIdentifier类，则应该将内容传递的字符串格式将被提取，或LanguageProfile类对象。

```
LanguageIdentifier object=new LanguageIdentifier(“this is english”)
```

下面给出的是Tika语言检测的示例程序。

```
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.language.LanguageIdentifier;

import org.xml.sax.SAXException;

public class LanguageDetection {

   public static void main(String args[])throws IOException, SAXExc

      LanguageIdentifier identifier = new LanguageIdentifier("this
      String language = identifier.getLanguage();
      System.out.println("Language of the given content is : " + la
   }
}
```

将以上代码保存为LanguageDetection.java并在命令提示符处使用以下命令运行它：

```
javac  LanguageDetection.java
java  LanguageDetection
```

它提供了以下的输出：

```
Language of the given content is : en
```

## 语言检测文档

要检测一个给定的文档的语言，必须使用parse()方法来解析它。parse()方法解析处理程序对象，这是传递给它的参数的内容，并将其存储。通过LanguageIdentifier类对象处理,构造函数的字符串的格式如下图所示：

```
parser.parse(inputstream, handler, metadata, context);
LanguageIdentifier object = new LanguageIdentifier(handler.toStrin
```

下面给出的是一个演示如何检测一个给定的文档的语言完整的程序：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.Parser;
import org.apache.tika.sax.BodyContentHandler;
import org.apache.tika.language.*;

import org.xml.sax.SAXException;

public class DocumentLanguageDetection {

   public static void main(final String[] args) throws IOException,

      //Instantiating a file object
      File file = new File("Example.txt");

      //Parser method parameters
      Parser parser = new AutoDetectParser();
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream content = new FileInputStream(file);

      //Parsing the given document
      parser.parse(content, handler, metadata, new ParseContext())

      LanguageIdentifier object = new LanguageIdentifier(handler.to
      System.out.println("Language name :" + object.getLanguage())
   }
}
```

将以上代码保存为SetMetadata.java并在命令提示符下运行：

```
javac  SetMetadata.java
java  SetMetadata
```

注意：假设sample.txt 有下列内容：

```
Hi students welcome to yiibai
```

它提供了以下的输出：

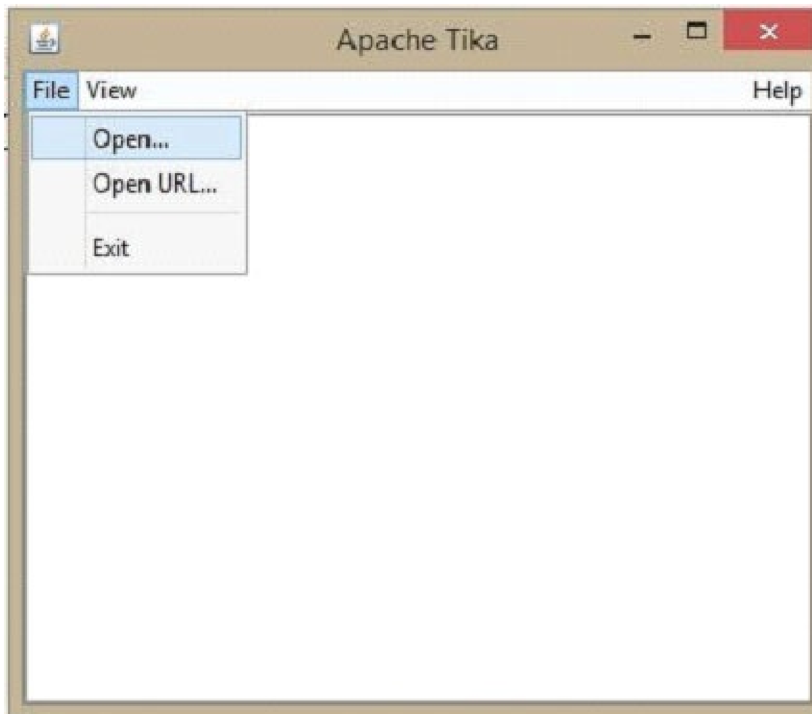
```
Language name :en
```

随着Tika jar，Tika提供了一个图形用户界面应用程序（GUI）和命令行界面（CLI）的应用程序。可以像其他Java应用程序在命令提示符下运行 Tika 的应用程序。

# TIKA图形界面/GUI - Tika教程

## 图形用户界面(GUI)

- Tika 提供了一个jar文件连同下面的链接它的源代码: http://tika.apache.org/download.htmll.
- 下载文件，并设置JAR类文件路径。
- 提取源代码zip文件夹，打开tika-app文件夹。
- 在解压缩文件夹“tika-1.6\tika-app\src\main\java\org\apache\Tika\gui” 后，会看到两个类文件：ParsingTransferHandler.java 和TikaGUI.java。
- 编译这两个类文件并执行TikaGUI.java类文件，它会打开下面的窗口。



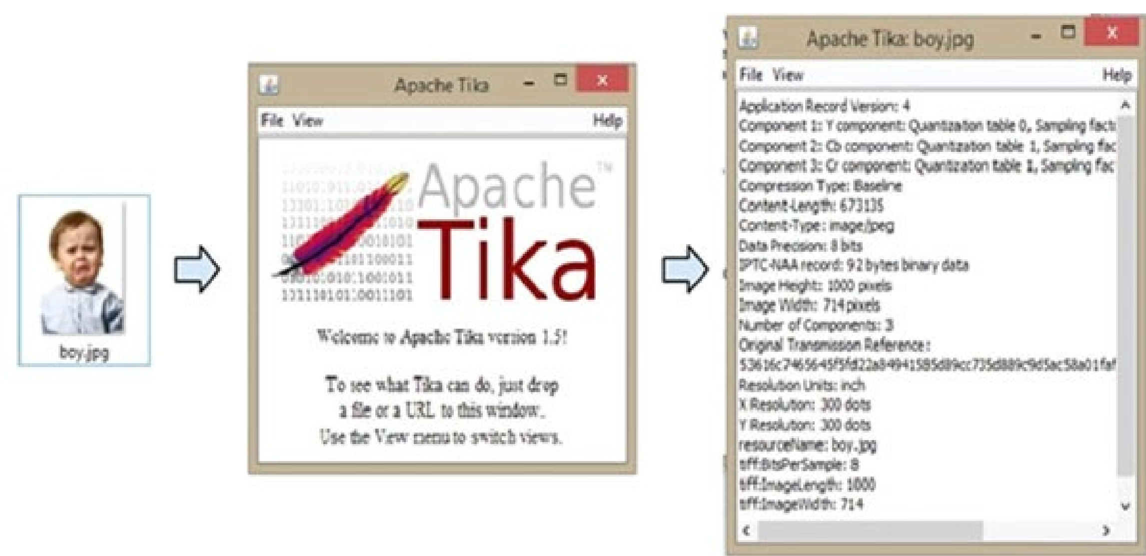

现在让我们看看如何利用Tika的GUI。

在GUI上，点击open打开，浏览并选择一个文件，该文件将被提取，或将其拖动到窗口的空白。

Tika 提取的文件的内容，并在五个不同的格式显示出来，即。元数据，格式化文本，纯文本，主要内容和结构化文本。可以选择任何想要的格式。

以同样的方式，会发现在“tika-1.6\tika-app\src\main\java\org\apache\tika\clil”文件夹中的CLI类。

下图显示了Tika能做到。当我们把图像托放在图形用户界面上，Tika提取并显示其元数据。


boy.jpg
Apache Tika
File View
Help
Apache
Tika
Welcome to Apache Tika version 1.5!
To see what Tika can do, just drop
a file or a URL to this window.
Use the View menu to switch views.
Apache Tika: boy.jpg
File View
Help
Application Record Version: 4
Component 1: Y component: Quantization table 0, Sampling fact
Component 2: Cb component: Quantization table 1, Sampling fac
Component 3: Cr component: Quantization table 1, Sampling fac
Compression Type: Baseline
Content-Length: 673135
Content-Type: image/jpeg
Data Precision: 8 bits
IPTC-NAA record: 92 bytes binary data
Image Height: 1000 pixels
Image Width: 714 pixels
Number of Components: 3
Original Transmission Reference:
Resolution Units: inch
X Resolution: 300 dots
Y Resolution: 300 dots
resourceName: boy.jpg
tiff:BitsPerSample: 8
tiff:ImageLength: 1000
tiff:ImageWidth: 714

# TIKA提取PDF - Tika教程

下面给出的程序是用来提取PDF文件内容和元数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.pdf.PDFParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class PdfParse {

   public static void main(final String[] args) throws IOException,

      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("E
      ParseContext pcontext = new ParseContext();

      //parsing the document using PDF parser
      PDFParser pdfparser = new PDFParser();
      pdfparser.parse(inputstream, handler, metadata,pcontext);

      //getting the content of the document
      System.out.println("Contents of the PDF :" + handler.toString

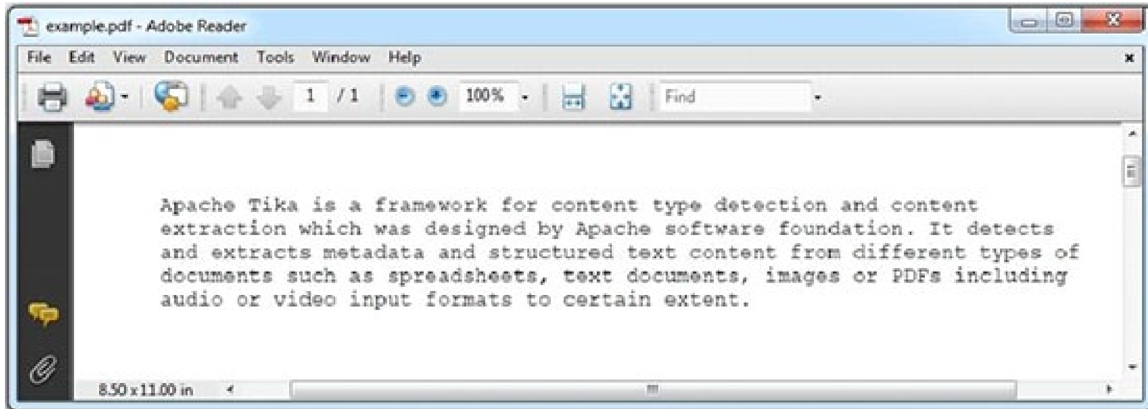
      //getting metadata of the document
      System.out.println("Metadata of the PDF:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name+ " : " + metadata.get(name));
      }
   }
}
```

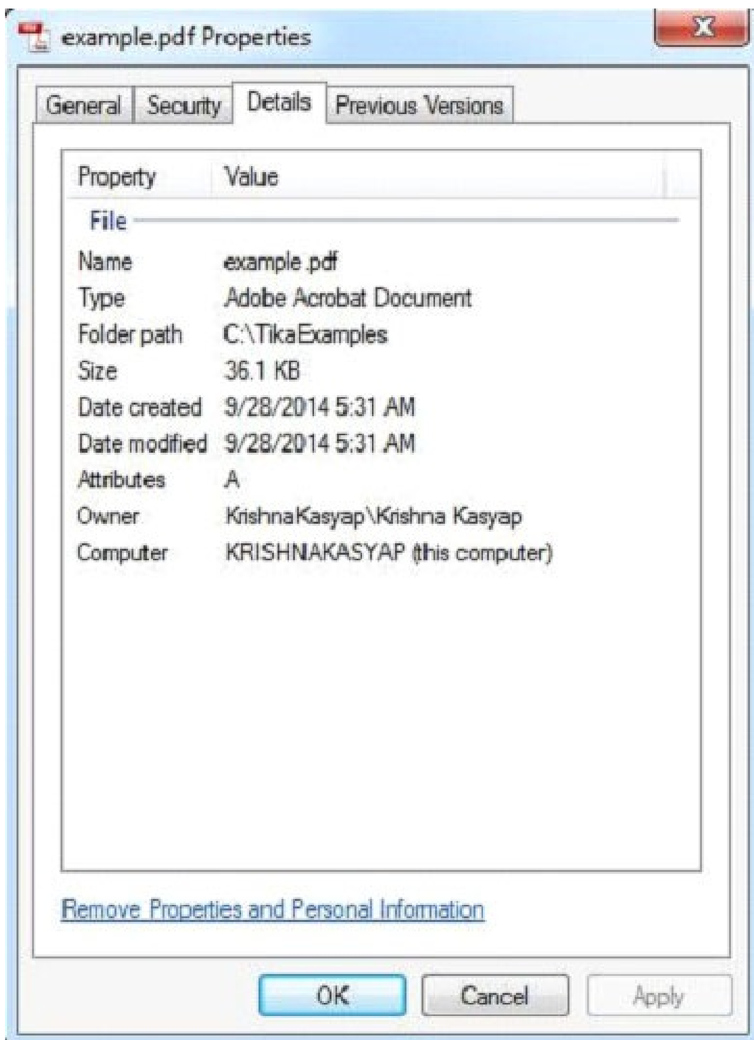
保存上述代码保存为PdfParse.java，并通过使用下面的命令从命令提示编译：

```
javac PdfParse.java
java PdfParse
```

下面给出的是Example.pdf文件的快照：




PDF文档具有以下属性：




执行上述程序后，会得到如下的输出

输出:

```
Contents of the PDF:

Apache Tika is a framework for content type detection and content e
which was designed by Apache software foundation. It detects and ex
and structured text content from different types of documents such
text documents, images or PDFs including audio or video input forma

Metadata of the PDF:

dcterms:modified :     2014-09-28T12:31:16Z
meta:creation-date :     2014-09-28T12:31:16Z
meta:save-date :     2014-09-28T12:31:16Z
dc:creator :     Krishna Kasyap
pdf:PDFVersion :     1.5
Last-Modified :     2014-09-28T12:31:16Z
Author :     Krishna Kasyap
dcterms:created :     2014-09-28T12:31:16Z
date :     2014-09-28T12:31:16Z
modified :     2014-09-28T12:31:16Z
creator :     Krishna Kasyap
xmpTPg:NPages :     1
Creation-Date :     2014-09-28T12:31:16Z
pdf:encrypted :     false
meta:author :     Krishna Kasyap
created :     Sun Sep 28 05:31:16 PDT 2014
dc:format :     application/pdf; version=1.5
producer :     Microsoft® Word 2013
Content-Type :     application/pdf
xmp:CreatorTool :     Microsoft® Word 2013
Last-Save-Date :     2014-09-28T12:31:16Z
```

# TIKA提取ODF - Tika教程

下面给出的是程序从打开Office文档格式(ODF)中提取内容和元数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.odf.OpenDocumentParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class OpenDocumentParse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("e
      ParseContext pcontext = new ParseContext();

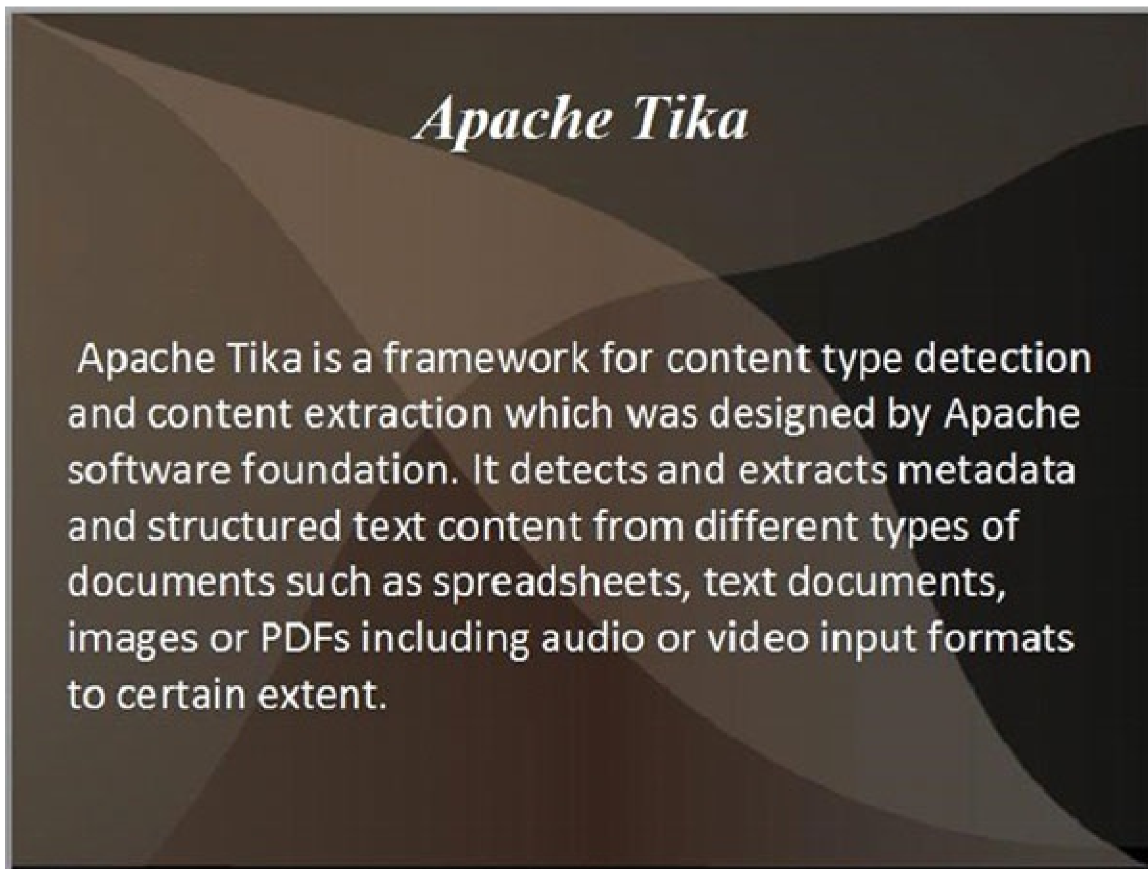
      //Open Document Parser
      OpenDocumentParser openofficeparser = new OpenDocumentParser
      openofficeparser.parse(inputstream, handler, metadata,pconte
      System.out.println("Contents of the document:" + handler.toS
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + " :  " + metadata.get(name));
      }
   }
}
```

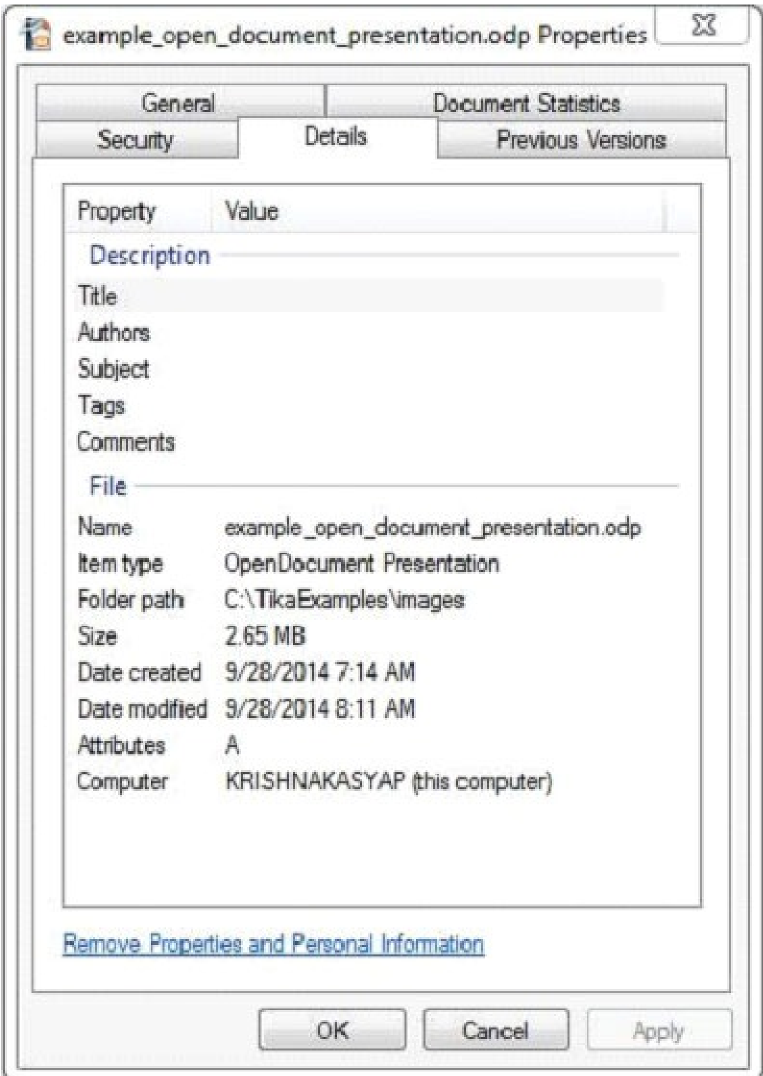
将以上代码保存为OpenDocumentParse.java，并通过使用以下命令，在命令提示符下编译：

```
javac OpenDocumentParse.java
java OpenDocumentParse
```

下面给出的是example_open_document_presentation.odp的快照：


Apache Tika
Apache Tika is a framework for content type detection and content extraction which was designed by Apache software foundation. It detects and extracts metadata and structured text content from different types of documents such as spreadsheets, text documents, images or PDFs including audio or video input formats to certain extent.

本文档具有以下属性：

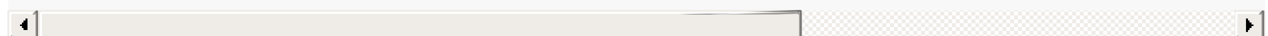



执行上述程序后，将得到下面的输出。

输出:

```
Contents of the document:
Apache Tika
Apache Tika is a framework for content type detection and content e
by Apache software foundation. It detects and extracts metadata and
different types of documents such as spreadsheets, text documents, 
or video input formats to certain extent.

Metadata of the document:
editing-cycles:   4
meta:creation-date:   2009-04-16T11:32:32.86
dcterms:modified:   2014-09-28T07:46:13.03
meta:save-date:   2014-09-28T07:46:13.03
Last-Modified:   2014-09-28T07:46:13.03
dcterms:created:   2009-04-16T11:32:32.86
date:   2014-09-28T07:46:13.03
modified:   2014-09-28T07:46:13.03
nbObject:   36
Edit-Time:   PT32M6S
Creation-Date:   2009-04-16T11:32:32.86
Object-Count:   36
meta:object-count:   36
generator:   OpenOffice/4.1.0$Win32 OpenOffice.org_project/410m18$E
Content-Type:   application/vnd.oasis.opendocument.presentation
Last-Save-Date:   2014-09-28T07:46:13.03
```



# TIKA提取MS Office文件 - Tika教程

下面给出的程序是用于从Microsoft Office文档中提取内容和元数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.microsoft.ooxml.OOXMLParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class MSExcelParse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("e
      ParseContext pcontext = new ParseContext();

      //OOXml parser
      OOXMLParser  msofficeparser = new OOXMLParser ();
      msofficeparser.parse(inputstream, handler, metadata,pcontext)
      System.out.println("Contents of the document:" + handler.toSt
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ": " + metadata.get(name));
      }
   }
}
```

保存上述代码保存为MSExelParse.java，并通过使用下面的命令从命令提示编译：

```
javac MSExcelParse.java
java MSExcelParse
```

下面给出的是 example_msExcel.xlsx 文件的快照

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Name | Age | Designation | Salary |
| 2 | Ramu | 50 | Manager | 50,000 |
| 3 | raheem | 40 | Assistant manager | 40,000 |
| 4 | Robert | 30 | Superviser | 30,000 |
| 5 | sita | 25 | Clerck | 25,000 |
| 6 | sameer | 25 | Section incharge | 20,000 |

给定的 Excel 文件具有如下性质 ：

example_msExcel.xlsx Properties

| Property | Value |
|---|---|
| Program name | Microsoft Excel |
| Company | |
| Manager | |
| Content created | 9/15/2006 5:00 PM |
| Date last saved | 9/28/2014 9:14 AM |
| Last printed | |
| **Content** | |
| Content status | |
| Content type | |
| Scale | No |
| Links dirty? | No |
| Language | |
| **File** | |
| Size | 8.01 KB |
| Date created | 9/28/2014 8:12 AM |
| Date modified | 9/28/2014 9:14 AM |
| Date accessed | 9/28/2014 9:14 AM |

执行上述程序后，将得到下面的输出。

输出:

```
Contents of the document:
Sheet1
Name    Age    Designation        Salary
Ramu    50    Manager            50,000
Raheem    40    Assistant manager    40,000
Robert    30    Superviser        30,000
sita    25    Clerk            25,000
sameer    25    Section in-charge    20,000

Metadata of the document:
meta:creation-date:    2006-09-16T00:00:00Z
dcterms:modified:    2014-09-28T15:18:41Z
meta:save-date:    2014-09-28T15:18:41Z
Application-Name:    Microsoft Excel
extended-properties:Company:
dcterms:created:    2006-09-16T00:00:00Z
Last-Modified:    2014-09-28T15:18:41Z
Application-Version:    15.0300
date:    2014-09-28T15:18:41Z
publisher:
modified:    2014-09-28T15:18:41Z
Creation-Date:    2006-09-16T00:00:00Z
extended-properties:AppVersion:    15.0300
protected:    false
dc:publisher:
extended-properties:Application:    Microsoft Excel
Content-Type:    application/vnd.openxmlformats-officedocument.spre
Last-Save-Date:    2014-09-28T15:18:41Z
```

# TIKA提取文本文档 - Tika教程

下面给出的程序是用来提取文本文档的内容和元数据：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.sax.BodyContentHandler;
import org.apache.tika.parser.txt.TXTParser;

import org.xml.sax.SAXException;

public class TextParser {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("e
      ParseContext pcontext=new ParseContext();

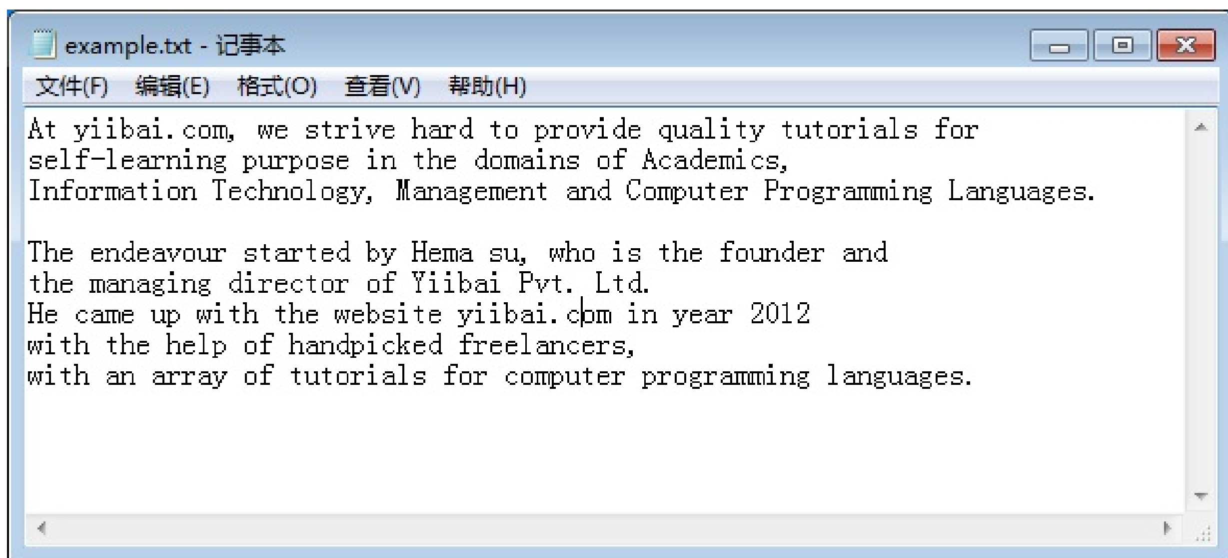
      //Text document parser
      TXTParser  TexTParser = new TXTParser();
      TexTParser.parse(inputstream, handler, metadata,pcontext);
      System.out.println("Contents of the document:" + handler.toSt
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + " : " + metadata.get(name));
      }
   }
}
```

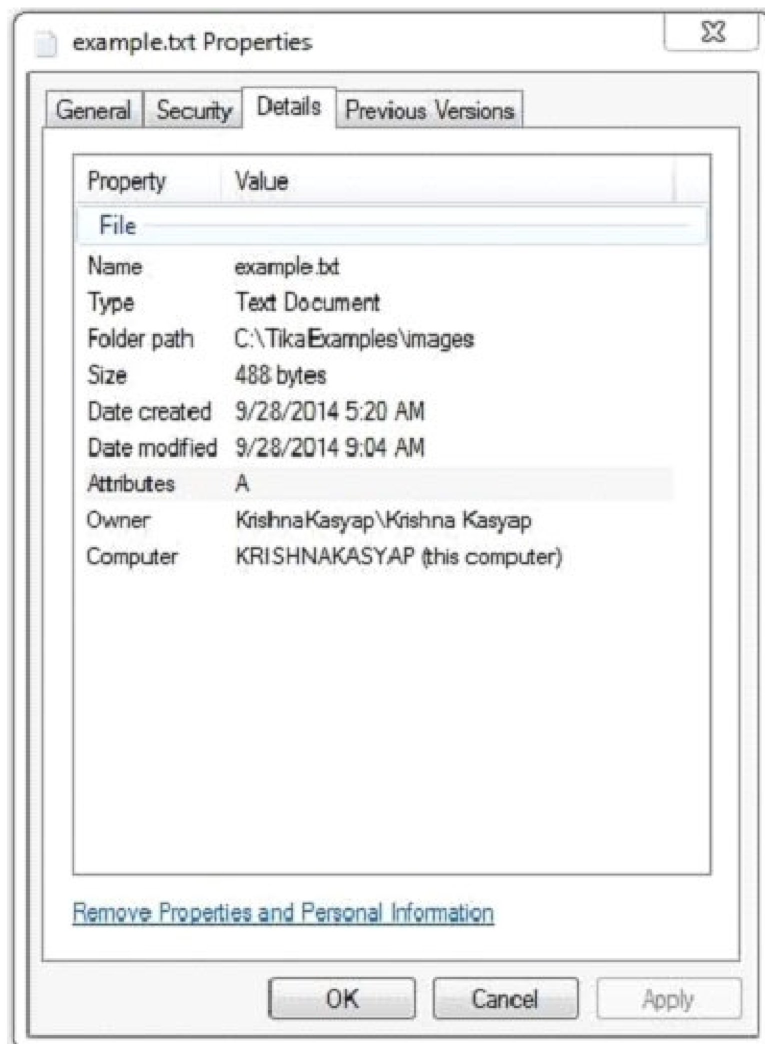
保存上述代码作为TextParser.java，并通过使用下面的命令从命令提示编译：

```
javac TextParser.java
java TextParser
```

下面给出的是example.txt文件的快照：




文本文件具有以下属性：




执行上述程序后，将得到下面的输出。

输出:

```
Contents of the document:
```

At Yiibai.com, we strive hard to provide quality tutorials for self-learning purpose in the domains of Academics, Information Technology, Management and Computer Programming Languages.

The endeavour started by Hema su, who is the founder and the managing director of Yiibai Pvt. Ltd. He came up with the website yiibai.com in year 2014 with the help of handpicked freelancers, with an array of tutorials for computer programming languages.

Metadata of the document: Content-Encoding: windows-1252 Content-Type: text/plain; charset=windows-1252

# TIKA提取HTML文档 - Tika教程

下面给出的是该程序用于从HTML文档提取内容和元数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.html.HtmlParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class HtmlParse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("e
      ParseContext pcontext = new ParseContext();

      //Html parser
      HtmlParser htmlparser = new HtmlParser();
      htmlparser.parse(inputstream, handler, metadata,pcontext);
      System.out.println("Contents of the document:" + handler.toSt
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ":   " + metadata.get(name));
      }
   }
}
```

保存上述代码保存为HtmlParse.java，并通过使用下面的命令从命令提示编译：

```
javac HtmlParse.java
java HtmlParse
```

下面给出的是 example.htmll 文档的快照。

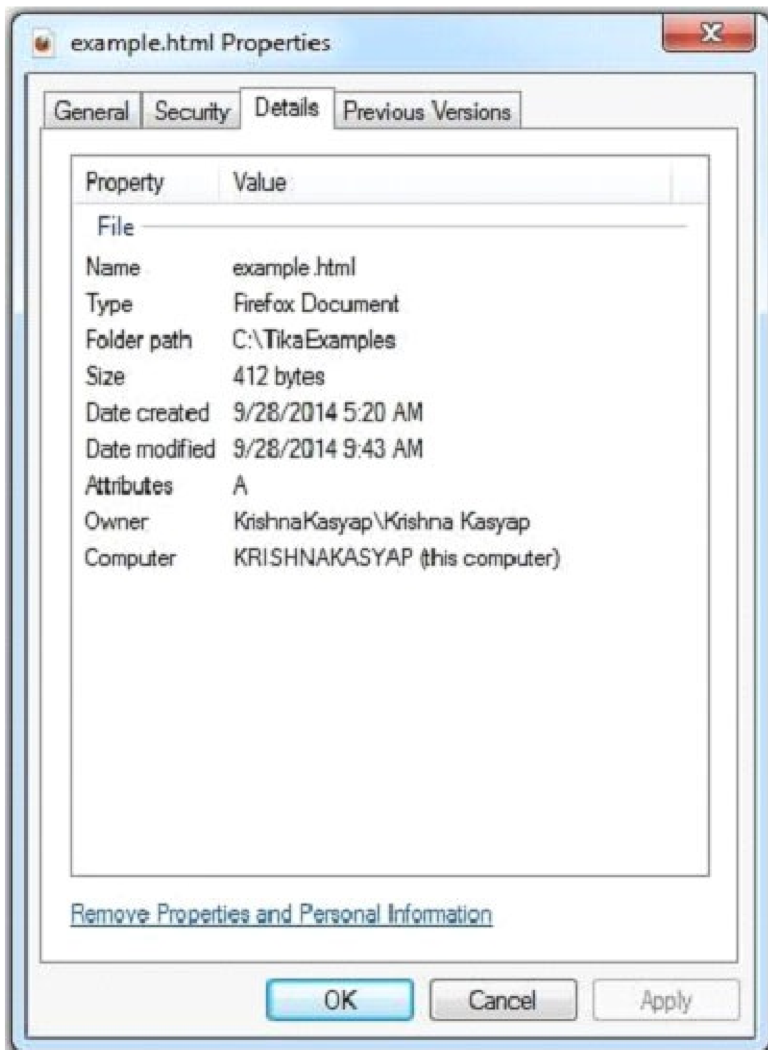
example1.html...

Home View

```
<tr>
<th>Name</th>
<th>Salary</th>
<th>age</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>50000</td>
<th>20</th>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>70000</td>
<th>25</th>
</tr>
<tr>
<td>umesh Raman</td>
<td>50000</td>
<th>30</th>
</tr>
</table>
</body>
</html>
```

100%

HTML文档有以下属性：




执行上述程序后，将得到下面的输出。

输出:

```
Contents of the document:
    Name                             Salary        age
    Ramesh Raman                     50000        20
    Shabbir Hussein                     70000           25
    Umesh Raman                     50000        30
    Somesh                              50000        35
Metadata of the document:
title:   HTML Table Header
Content-Encoding:   windows-1252
Content-Type:   text/html; charset=windows-1252
dc:title:   HTML Table Header
```

# TIKA提取XML文档 - Tika教程

下面给出的程序是用来从XML文档提取内容和元数据：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.xml.XMLParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class XmlParse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("
      ParseContext pcontext = new ParseContext();

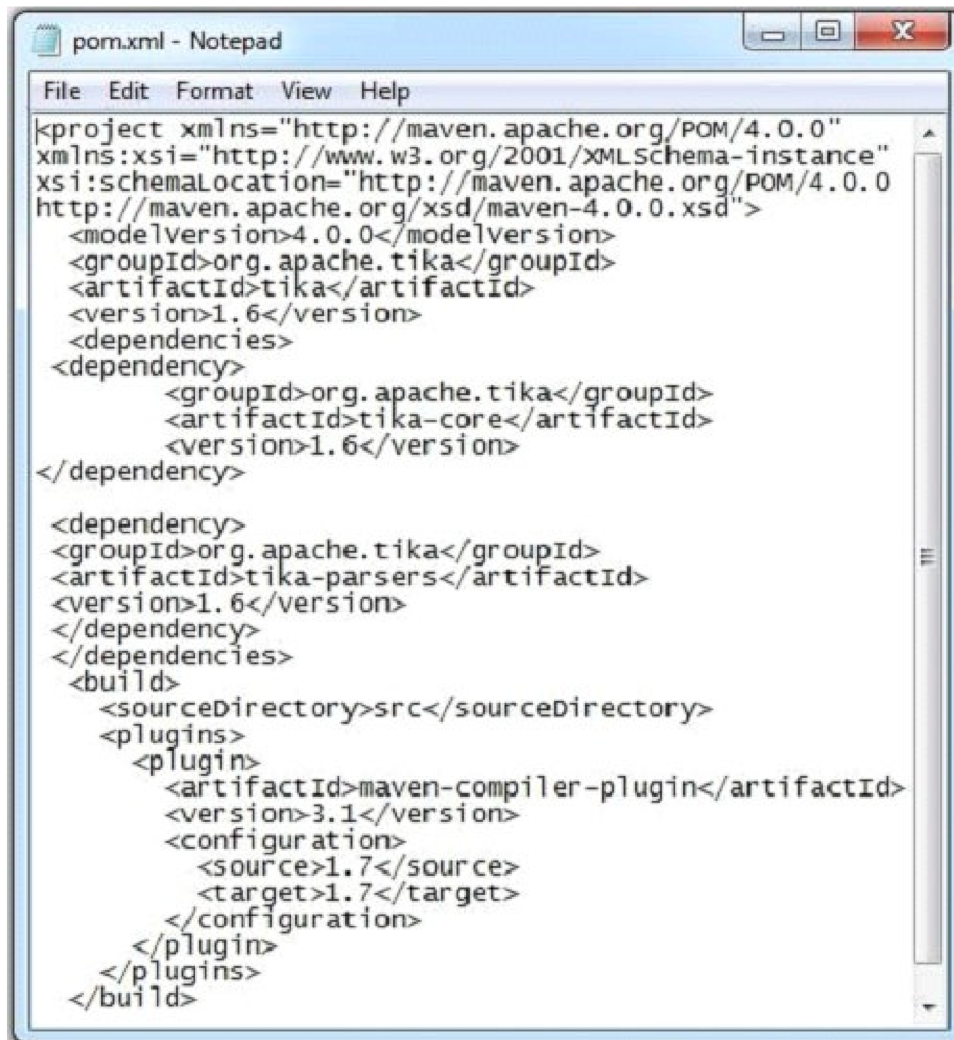
      //Xml parser
      XMLParser xmlparser = new XMLParser();
      xmlparser.parse(inputstream, handler, metadata, pcontext);
      System.out.println("Contents of the document:" + handler.toS
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ": " + metadata.get(name));
      }
   }
}
```

保存上述代码保存为XmlParse.java，并通过使用下面的命令从命令提示编译：

```
javac XmlParse.java
java XmlParse
```

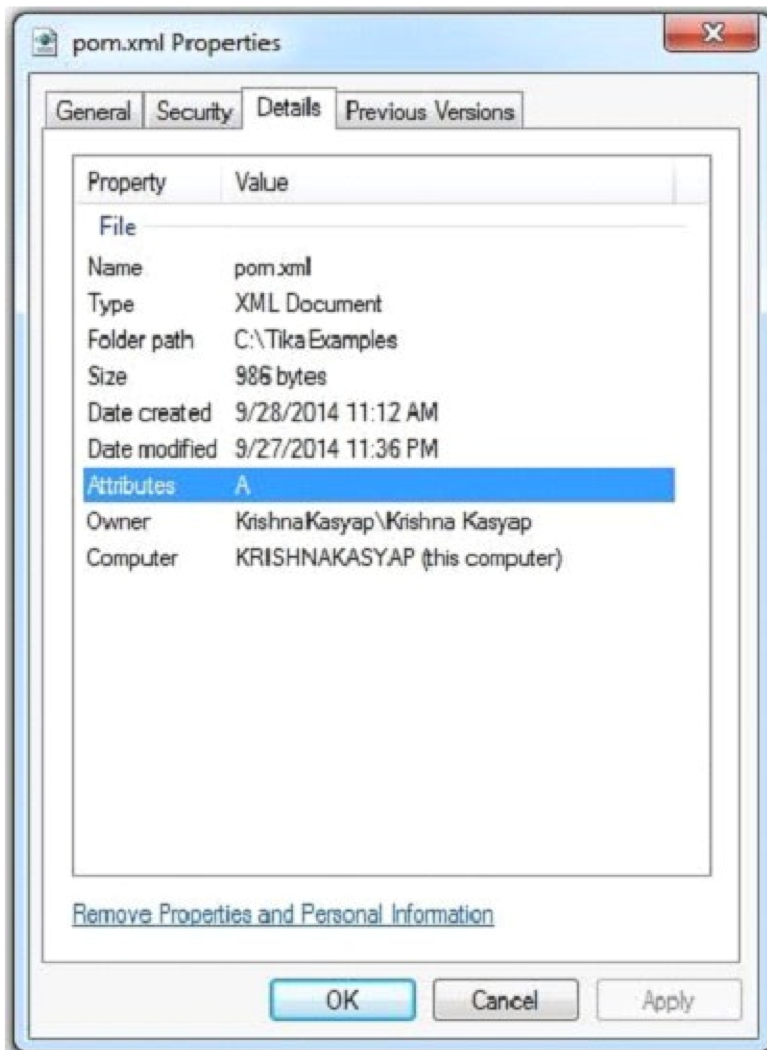
下面给出的是pom.xml文件的快照



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.apache.tika</groupId>
  <artifactId>tika</artifactId>
  <version>1.6</version>
  <dependencies>
 <dependency>
        <groupId>org.apache.tika</groupId>
        <artifactId>tika-core</artifactId>
        <version>1.6</version>
</dependency>

 <dependency>
 <groupId>org.apache.tika</groupId>
 <artifactId>tika-parsers</artifactId>
 <version>1.6</version>
 </dependency>
 </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

本文档具有以下属性：




执行上述程序后，将得到下面的输出。

输出:

```
Contents of the document:
4.0.0
org.apache.tika
tika
1.6
org.apache.tika
tika-core
1.6
org.apache.tika
tika-parsers
1.6
src
maven-compiler-plugin
3.1
1.7
1.7

Metadata of the document:
Content-Type:   application/xml
```

# TIKA提取.class文件 - Tika教程

下面给出的是该程序提取.class文件的内容和元数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.asm.ClassParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class JavaClassParse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("E
      ParseContext pcontext = new ParseContext();

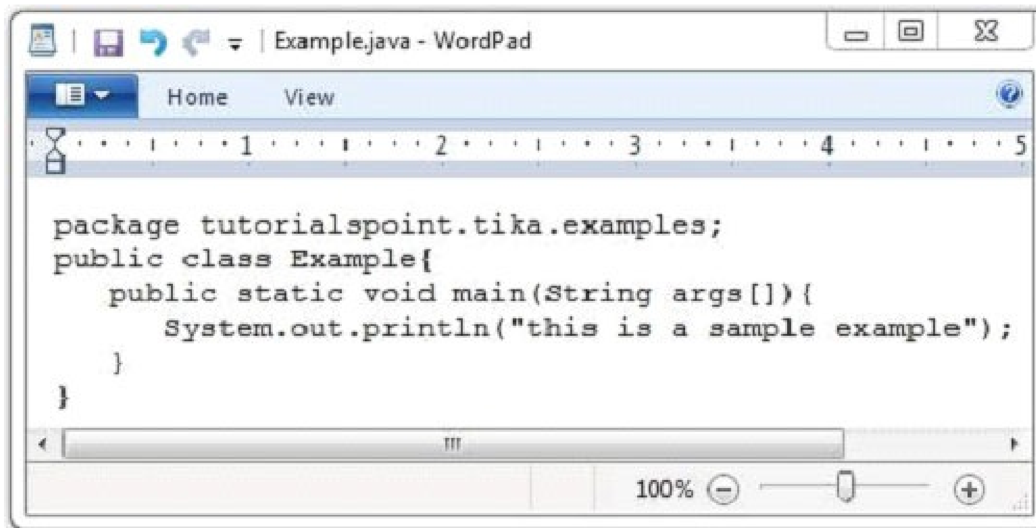
      //Html parser
      ClassParser  ClassParser = new  ClassParser();
      ClassParser.parse(inputstream, handler, metadata,pcontext);
      System.out.println("Contents of the document:" + handler.toSt
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + " :  " + metadata.get(name));
      }
   }
}
```

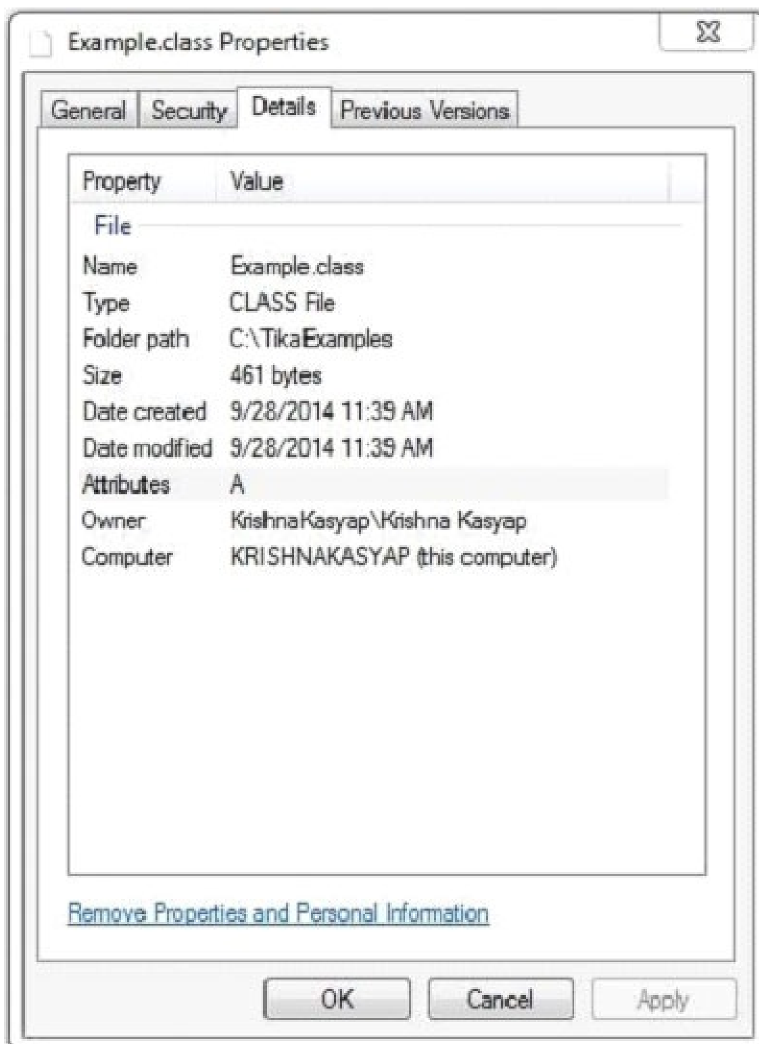
保存上述代码作为JavaClassParse.java，并通过使用下面的命令从命令提示编译：

```
javac JavaClassParse.java
java JavaClassParse
```

下面给出的是Example.java编译执行后，将得到example.class文件的快照：




此.class文件有以下属性：




执行上述程序后，将得到下面的输出。

输出:

```
Contents of the document:
package yiibai.tika.examples;
public synchronized class Example {
   public void Example();
   public static void main(String[]);
}

Metadata of the document:
title: Example
resourceName: Example.class
dc:title: Example
```

# TIKA提取JAR文件 - Tika教程

下面给出的程序是用来从一个Java存档(JAR)文件提取内容和元数据：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.sax.BodyContentHandler;
import org.apache.tika.parser.pkg.PackageParser;

import org.xml.sax.SAXException;

public class PackageParse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("E
      ParseContext pcontext = new ParseContext();

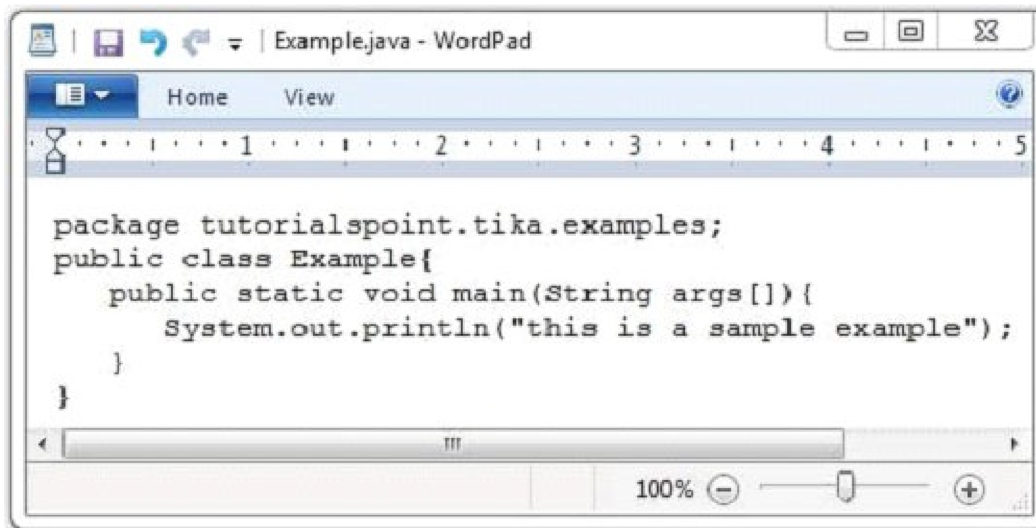
      //Package parser
      PackageParser packageparser = new PackageParser();
      packageparser.parse(inputstream, handler, metadata,pcontext);
      System.out.println("Contents of the document: " + handler.toS
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ":   " + metadata.get(name));
      }
   }
}
```

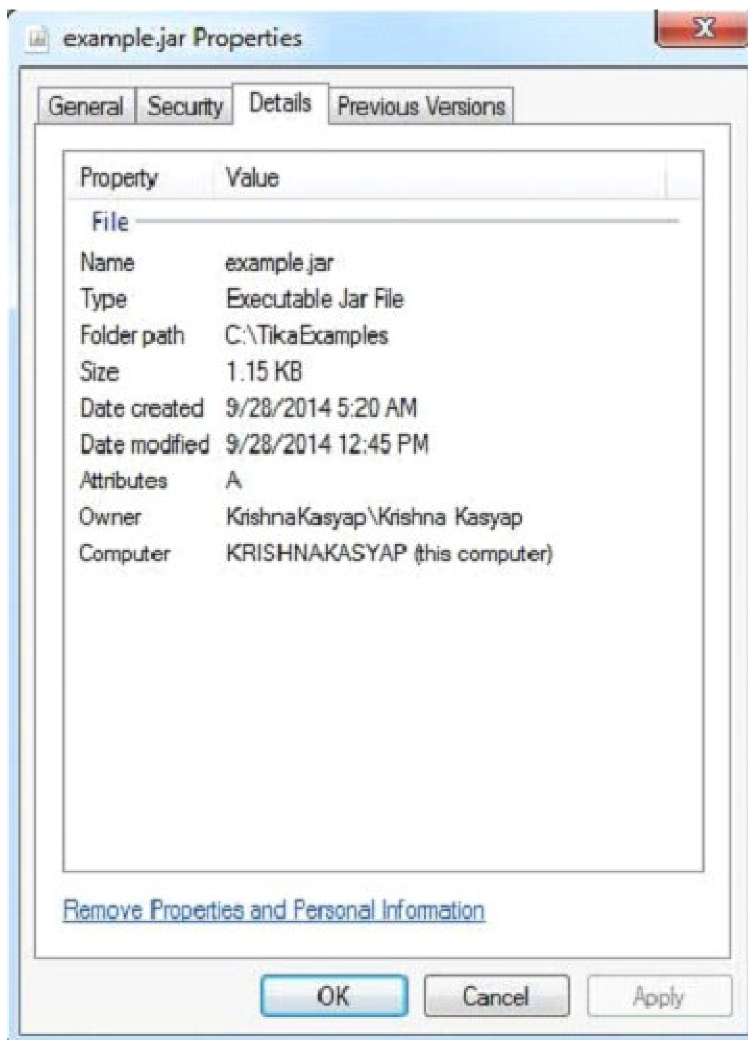
保存上述代码为PackageParse.java，并通过使用下面的命令从命令提示编译：

```
javac PackageParse.java
java PackageParse
```

在这里，我们通过下面的Example.java文件得到jar文件。




jar文件具有以下属性：




执行上述程序后，将得到下面的输出。

输出:

```
Contents of the document:
META-INF/MANIFEST.MF
yiibai/tika/examples/Example.class

Metadata of the document:
Content-Type:   application/zip
```

# TIKA提取图像文件 - Tika教程

下面给出的该程序是从一个JPEG图像中提取的内容和元数据。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.jpeg.JpegParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class JpegParse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("b
      ParseContext pcontext = new ParseContext();

      //Jpeg Parse
      JpegParser  JpegParser = new JpegParser();
      JpegParser.parse(inputstream, handler, metadata,pcontext);
      System.out.println("Contents of the document:" + handler.toSt
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ": " + metadata.get(name));
      }
   }
}
```

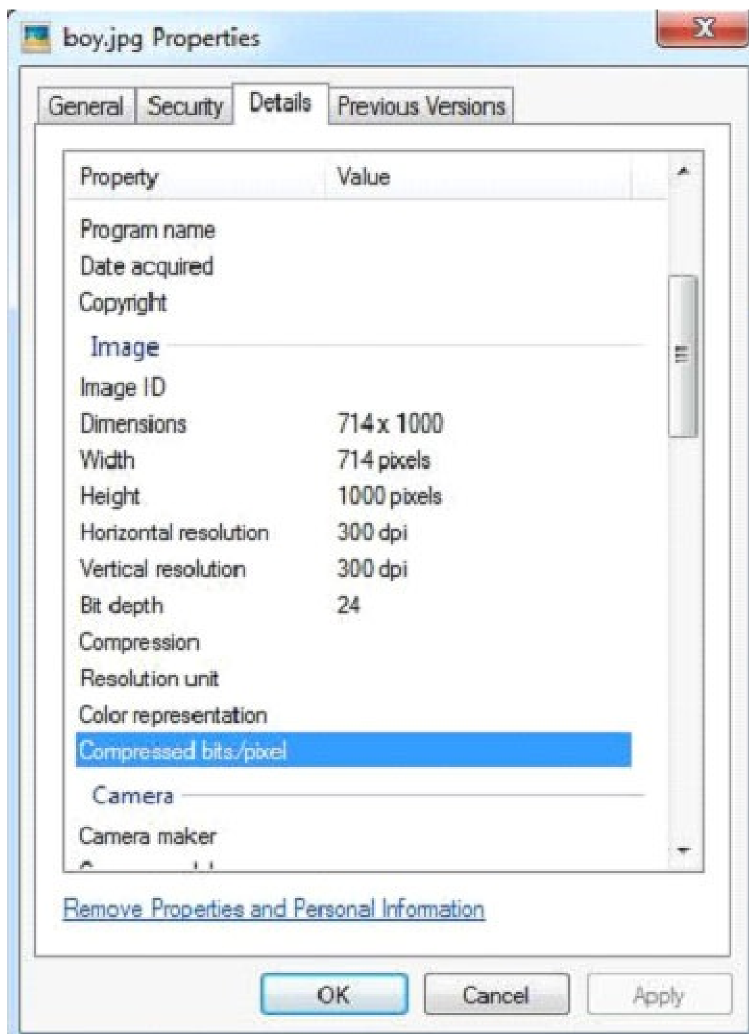
保存上述代码为JpegParse.java，并通过使用下面的命令从命令提示编译：

```
javac JpegParse.java
java JpegParse
```

在这里，我们通过下面的example.jpeg文件：




JPEG文件具有以下属性：




在执行上面的应用程序，会得到如下的输出。

输出:

```
Contents of the document:

Metadata of the document:
IPTC-NAA record: 92 bytes binary data
Number of Components: 3
Image Height: 1000 pixels
Resolution Units: inch
Data Precision: 8 bits
tiff:BitsPerSample: 8
Compression Type: Baseline
Component 1: Y component: Quantization table 0, Sampling factors 1
Component 2: Cb component: Quantization table 1, Sampling factors 1
tiff:ImageLength: 1000
Component 3: Cr component: Quantization table 1, Sampling factors 1
X Resolution: 300 dots
tiff:ImageWidth: 714
Application Record Version: 4
Image Width: 714 pixels
Original Transmission Reference:
 53616c7465645f5fd22a84941585d89cc735d889c9d5ac58a01faf2c92ee3c6f9b
Y Resolution: 300 dots
```

# TIKA提取mp4文件 - Tika教程

下面给出的程序是用来从mp4文件提取内容和元数据：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.mp4.MP4Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class Mp4Parse {

   public static void main(final String[] args) throws IOException,

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("e
      ParseContext pcontext = new ParseContext();

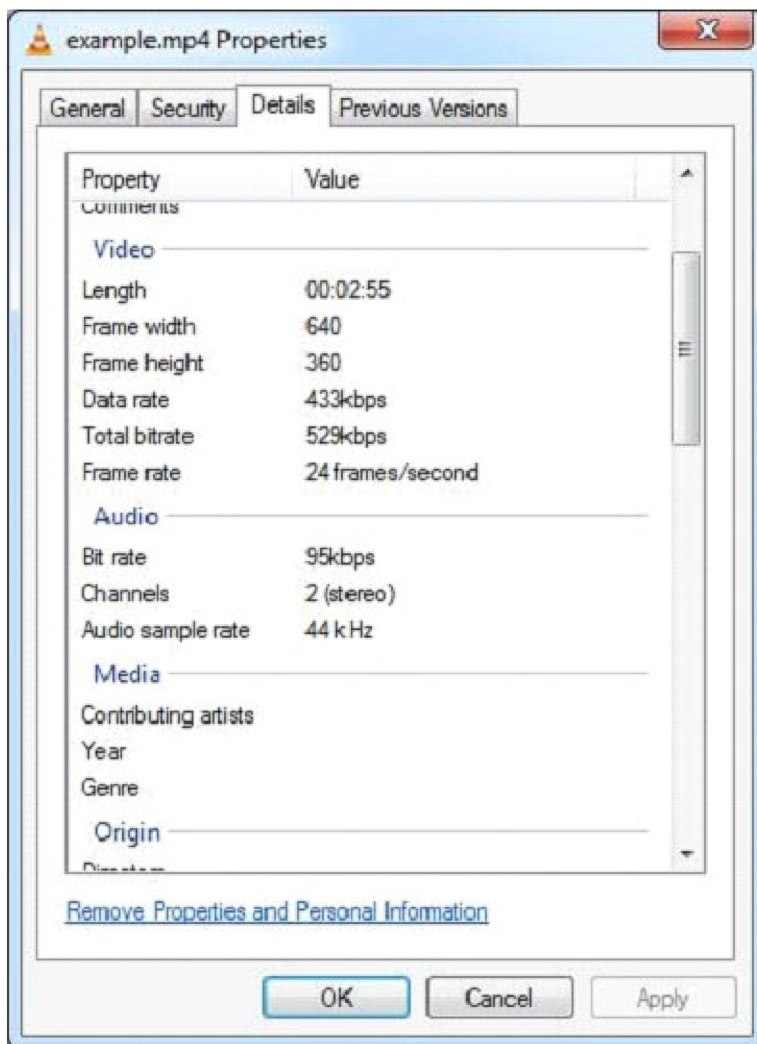
      //Html parser
      MP4Parser MP4Parser = new MP4Parser();
      MP4Parser.parse(inputstream, handler, metadata,pcontext);
      System.out.println("Contents of the document:  :" + handler.t
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ": " + metadata.get(name));
      }
   }
}
```

保存上述代码保存为JpegParse.java，并通过使用下面的命令从命令提示编译：

```
javac Mp4Parse.java
java Mp4Parse
```

下面给出的是example.mp4的快照属性：




执行上述程序后，得到如下的输出

输出:

```
Contents of the document:

Metadata of the document:
dcterms:modified: 2014-01-06T12:10:27Z
meta:creation-date: 1904-01-01T00:00:00Z
meta:save-date: 2014-01-06T12:10:27Z
Last-Modified: 2014-01-06T12:10:27Z
dcterms:created: 1904-01-01T00:00:00Z
date: 2014-01-06T12:10:27Z
tiff:ImageLength: 360
modified: 2014-01-06T12:10:27Z
Creation-Date: 1904-01-01T00:00:00Z
tiff:ImageWidth: 640
Content-Type: video/mp4
Last-Save-Date: 2014-01-06T12:10:27Z
```

# TIKA提取MP3文件 - Tika教程

下面给出的程序是用于提取MP3文件内容和元数据：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.mp3.LyricsHandler;
import org.apache.tika.parser.mp3.Mp3Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class Mp3Parse {

   public static void main(final String[] args) throws Exception, I

      //detecting the file type
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(new File("e
      ParseContext pcontext = new ParseContext();

      //Mp3 parser
      Mp3Parser  Mp3Parser = new  Mp3Parser();
      Mp3Parser.parse(inputstream, handler, metadata, pcontext);
      LyricsHandler lyrics = new LyricsHandler(inputstream,handler)

      while(lyrics.hasLyrics()) {
         System.out.println(lyrics.toString());
      }

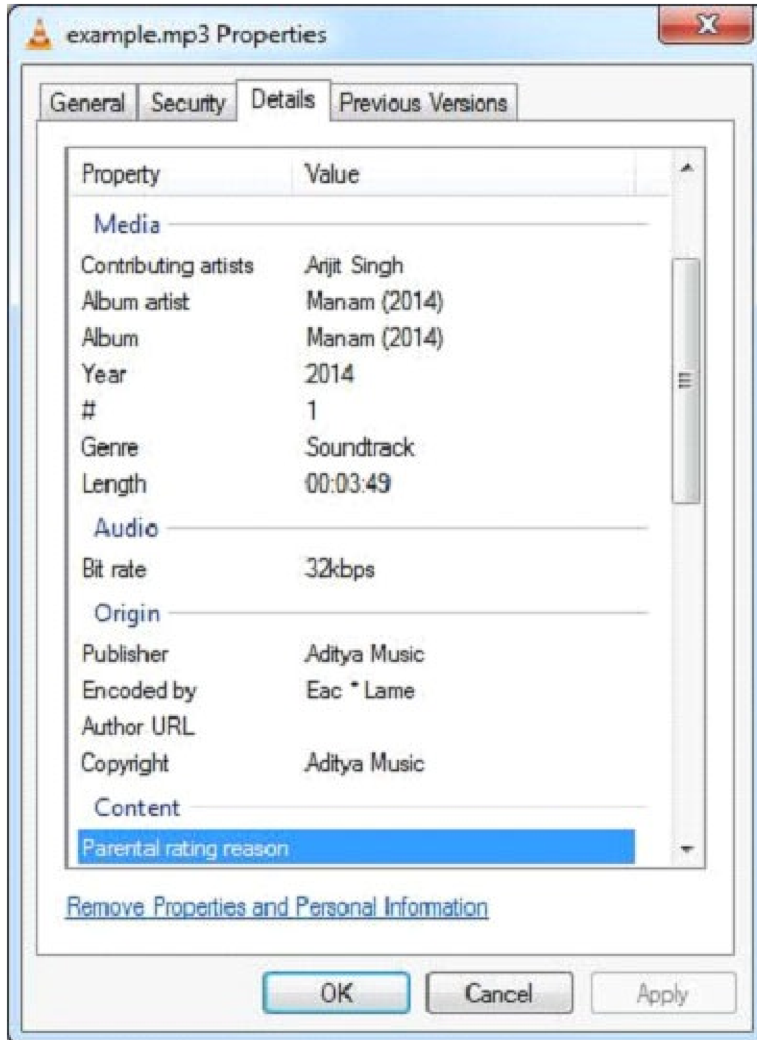
      System.out.println("Contents of the document:" + handler.toSt
      System.out.println("Metadata of the document:");
      String[] metadataNames = metadata.names();

      for(String name : metadataNames) {
         System.out.println(name + ": " + metadata.get(name));
      }
   }
}
```

保存上述代码保存为JpegParse.java，并通过使用下面的命令从命令提示编译：

```
javac Mp3Parse.java
java Mp3Parse
```

在这里，我们传递一个example.mp3文件具有以下属性：




执行上述程序后，将得到下面的输出。

如果给定的文件有任何的歌词，我们的应用程序将捕获和显示，和输出。

输出:

```
Contents of the document:
Kanulanu Thaake
Arijit Singh
Manam (2014), track 01/06
2014
Soundtrack
30171.65
eng -
DRGM
Arijit Singh
Manam (2014), track 01/06
2014
Soundtrack
30171.65
eng -
DRGM

Metadata of the document:
xmpDM:releaseDate: 2014
xmpDM:duration: 30171.650390625
xmpDM:audioChannelType: Stereo
dc:creator: Arijit Singh
xmpDM:album: Manam (2014)
Author: Arijit Singh
xmpDM:artist: Arijit Singh
channels: 2
xmpDM:audioSampleRate: 44100
xmpDM:logComment: eng -
DRGM
xmpDM:trackNumber: 01/06
version: MPEG 3 Layer III Version 1
creator: Arijit Singh
xmpDM:composer: Music : Anoop Rubens | Lyrics : Vanamali
xmpDM:audioCompressor: MP3
title: Kanulanu Thaake
samplerate: 44100
meta:author: Arijit Singh
xmpDM:genre: Soundtrack
Content-Type: audio/mpeg
xmpDM:albumArtist: Manam (2014)
dc:title: Kanulanu Thaake
```

# XStream教程

XStream是一个简单的基于Java库，Java对象序列化到XML，反之亦然(即：可以轻易的将Java对象和xml文档相互转换)。

## 特点

- 使用方便 - XStream的API提供了一个高层次外观，以简化常用的用例。
- 无需创建映射 - XStream的API提供了默认的映射大部分对象序列化。
- 性能 - XStream快速和低内存占用，适合于大对象图或系统。
- 干净的**XML** - XStream创建一个干净和紧凑XML结果，这很容易阅读。
- 不需要修改对象 - XStream可序列化的内部字段，如私有和最终字段，支持非公有制和内部类。默认构造函数不是强制性的要求。
- 完整对象图支持 - XStream允许保持在对象模型中遇到的重复引用，并支持循环引用。
- 可自定义的转换策略 - 定制策略可以允许特定类型的定制被表示为XML的注册。
- 安全框架 - XStream提供了一个公平控制有关解组的类型，以防止操纵输入安全问题。
- 错误消息 - 出现异常是由于格式不正确的XML时，XStream抛出一个统一的例外，提供了详细的诊断，以解决这个问题。
- 另一种输出格式 - XStream支持其它的输出格式，如JSON。

## 常见的用途

- 传输
- 持久化
- 配置
- 单元测试

# XStream环境设置 - XStream教程

## 本地环境设置

如果愿意设置Java编程语言环境，那么这部分指导如何下载和设置Java在机器上。请按照以下步骤来设置环境。

Java SE是免费的，提供的链接下载Java。所以，根据操作系统版本下载。

按照说明下载java并运行.exe到机器上安装Java。机器上安装了Java以后，还需要设置环境变量指向正确的安装目录：

## 为Windows 2000/XP设置路径：

假设安装在c:Program Filesjavajdk目录：

- 在“我的电脑”右键单击并选择“属性”。
- 在“高级”选项卡下单击“环境变量”按钮。
- 现在，改变“**Path**”变量，因此，它也包含了Java可执行文件的路径。例如，如果路径当前设置为“C:WINDOWSSYSTEM32”，然后更改您的路径如“'C:WINDOWSSYSTEM32;c:Program Filesjavajdkin”。

## 为Linux，UNIX，Solaris和FreeBSD设置Java路径：

环境变量PATH应设置为指向已安装的Java二进制文件的位置。请参考shell文件。

例如，如果使用bash作为shell，那么将下面的行添加到'.bashrc'文件的结尾：export PATH=/path/to/java:$PATH'

## 流行的Java编辑器：

编写Java程序，需要一个文本编辑器。在市场上有很多可用更复杂的IDE。现在，可以考虑下列之一：

- **Notepad:** 在Windows机器上，可以使用像记事本的任何简单的文本编辑器（推荐本教程），TextPad。
- **Netbeans:** 是一个Java IDE，它是开源和免费的，可从以下地址下载http://www.netbeans.org/index.html.
- **Eclipse:** 也是一个Java IDE由Eclipse开源社区开发，可以从下载http://www.eclipse.org/.

# 下载XStream的归档文件

下载最新的版本XStream的jar文件从 xstream-1.4.7.jar，在写这篇教程的时候，下载的是 XStream-1.4.7.jar 并将其复制到 C：> XStream 文件夹中。

| OS | 归档名称 |
|---|---|
| Windows | xstream-1.4.7.jar |
| Linux | xstream-1.4.7.jar |
| Mac | xstream-1.4.7.jar |

# 设置XStream环境

将用xstream_home环境变量指向 Guava jar 存储在计算机上的基本目录位置。假设，我们已经提取XStream-1.4.7.jar在各种操作系统XStream文件夹，如下所示。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 XStream_HOME 为 C:XStream |
| Linux | export XStream_HOME=/usr/local/XStream |
| Mac | export XStream_HOME=/Library/XStream |

# 设置CLASSPATH变量

设置 **CLASSPATH** 环境变量指向 XStream jar 位置。假设，我们已经存储 XStream-1.4.7.jar 在 XStream 文件夹的不同的操作系统如下。

| OS | 输出 |
|---|---|
| Windows | 设置环境变量 CLASSPATH 为 %CLASSPATH%;%XStream_HOME%xstream-1.4.7.jar;.; |
| Linux | export CLASSPATH=$CLASSPATH:$XStream_HOME/xstream-1.4.7.jar:. |
| Mac | export CLASSPATH=$CLASSPATH:$XStream_HOME/xstream-1.4.7.jar:. |

# XStream入门应用程序 - XStream教程

在进入XStream库的细节之前，让我们来看看应用程序操作。在这个例子中，我们创建Student和Address类。还将创建一个Student对象，然后将其序列化到一个XML字符串。然后反序列化的同一个XML字符串，以重新获得学生对象。

创建一个名为XStreamTester的Java类文件在 C:\>XStream_WORKSPACE.

文件: *XStreamTester.java*

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.stream.StreamResult;

import org.xml.sax.InputSource;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.StaxDriver;

public class XStreamTester {
   public static void main(String args[]){
      XStreamTester tester = new XStreamTester();
      XStream xstream = new XStream(new StaxDriver());

      Student student = tester.getStudentDetails();

      //Object to XML Conversion
      String xml = xstream.toXML(student);
      System.out.println(formatXml(xml));

      //XML to Object Conversion
      Student student1 = (Student)xstream.fromXML(xml);
      System.out.println(student1);
   }

   private Student getStudentDetails(){
      Student student = new Student();
      student.setFirstName("Mahesh");
      student.setLastName("Parashar");
      student.setRollNo(1);
      student.setClassName("1st");

      Address address = new Address();
```

```
      address.setArea("H.No. 16/3, Preet Vihar.");
      address.setCity("Delhi");
      address.setState("Delhi");
      address.setCountry("India");
      address.setPincode(110012);

      student.setAddress(address);
      return student;
   }

   public static String formatXml(String xml){
      try{
         Transformer serializer= SAXTransformerFactory.newInstance(
         serializer.setOutputProperty(OutputKeys.INDENT, "yes");
         serializer.setOutputProperty("{http://xml.apache.org/xslt]
         Source xmlSource=new SAXSource(new InputSource(new ByteArr
         StreamResult res =  new StreamResult(new ByteArrayOutputSt
         serializer.transform(xmlSource, res);
         return new String(((ByteArrayOutputStream)res.getOutputStr
      }catch(Exception e){
         return xml;
      }
   }
}

class Student {
   private String firstName;
   private String lastName;
   private int rollNo;
   private String className;
   private Address address;

   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName(String lastName) {
      this.lastName = lastName;
   }
   public int getRollNo() {
      return rollNo;
   }
   public void setRollNo(int rollNo) {
      this.rollNo = rollNo;
   }
   public String getClassName() {
      return className;
   }
```

```
   public void setClassName(String className) {
      this.className = className;
   }
   public Address getAddress() {
      return address;
   }
   public void setAddress(Address address) {
      this.address = address;
   }

   public String toString(){
      StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append("Student [ ");
      stringBuilder.append("\nfirstName: ");
      stringBuilder.append(firstName);
      stringBuilder.append("\nlastName: ");
      stringBuilder.append(lastName);
      stringBuilder.append("\nrollNo: ");
      stringBuilder.append(rollNo);
      stringBuilder.append("\nclassName: ");
      stringBuilder.append(className);
      stringBuilder.append("\naddress: ");
      stringBuilder.append(address);
      stringBuilder.append(" ]");
      return stringBuilder.toString();
   }
}

class Address {
   private String area;
   private String city;
   private String state;
   private String country;
   private int pincode;

   public String getArea() {
      return area;
   }
   public void setArea(String area) {
      this.area = area;
   }
   public String getCity() {
      return city;
   }
   public void setCity(String city) {
      this.city = city;
   }
   public String getState() {
      return state;
   }
   public void setState(String state) {
      this.state = state;
   }
```

```
    public String getCountry() {
       return country;
    }
    public void setCountry(String country) {
       this.country = country;
    }
    public int getPincode() {
       return pincode;
    }
    public void setPincode(int pincode) {
       this.pincode = pincode;
    }
    public String toString(){
       StringBuilder stringBuilder = new StringBuilder();
       stringBuilder.append("\nAddress [ ");
       stringBuilder.append("\narea: ");
       stringBuilder.append(area);
       stringBuilder.append("\ncity: ");
       stringBuilder.append(city);
       stringBuilder.append("\nstate: ");
       stringBuilder.append(state);
       stringBuilder.append("\ncountry: ");
       stringBuilder.append(country);
       stringBuilder.append("\npincode: ");
       stringBuilder.append(pincode);
       stringBuilder.append(" ]");
       return stringBuilder.toString();
    }
}
```

验证结果

使用javac编译器编译如下类：

```
C:\XStream_WORKSPACE>javac XStreamTester.java
```

现在运行XStreamTester看到结果：

```
C:\XStream_WORKSPACE>java XStreamTester
```

验证输出

```
<?xml version="1.0" encoding="UTF-8"?>
<Student>
  <firstName>Mahesh</firstName>
  <lastName>Parashar</lastName>
  <rollNo>1</rollNo>
  <className>1st</className>
  <address>
    <area>H.No. 16/3, Preet Vihar.</area>
    <city>Delhi</city>
    <state>Delhi</state>
    <country>India</country>
    <pincode>110012</pincode>
  </address>
</Student>

Student [
firstName: Mahesh
lastName: Parashar
rollNo: 1
className: 1st
address:
Address [
area: H.No. 16/3, Preet Vihar.
city: Delhi
state: Delhi
country: India
pincode: 110012 ] ]
```

# 记住以下步骤

以下是这里要考虑的重要步骤。

## 第1步：创建XStream对象。

通过它传递一个StaxDriver创建XStream对象。StaxDriver使用SAX解析器(可从Java6)，一个快速的XML解析器。

```
XStream xstream = new XStream(new StaxDriver());
```

## 第2步：序列化对象到XML。

使用toXML() 方法来获取对象的XML字符串表示。

```
//Object to XML Conversion
String xml = xstream.toXML(student);
```

## 第3步：反序列化XML获得对象。

使用 fromXML()方法来从XML对象。

```
//XML to Object Conversion
Student student1 = (Student)xstream.fromXML(xml);
```

# XStream混叠 - XStream教程

混叠是一种技术来定制生成XML或者使用XStream特定的格式化XML。假设，一个下面的XML格式是用于序列化/反序列化Student对象。

```
<student name="Suresh">
   <note>
   <title>first</title>
      <description>My first assignment.</description>
   </note>
   <note>
      <title>second</title>
      <description>My second assignment.</description>
   </note>
</student>
```

根据上面的XML格式，让我们创建的模型类。

```
class Student {
   private String studentName;
   private List<Note> notes = new ArrayList<Note>();

   public Student(String name) {
      this.studentName = name;
   }
   public void addNote(Note note) {
      notes.add(note);
   }
   public String getName(){
      return studentName;
   }
   public List<Note> getNotes(){
      return notes;
   }
}

class Note {
   private String title;
   private String description;

   public Note(String title, String description) {
      this.title = title;
      this.description = description;
   }

   public String getTitle(){
      return title;
   }

   public String getDescription(){
      return description;
   }
}
```

我们使用 XStream 测试上述对象序列化。

创建一个Java类名为XStreamTester在文件 C:\>XStream_WORKSPACE\com\yiibai\xstream.

*File: XStreamTester.java*

```
package com.yiibai.xstream;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.util.ArrayList;
import java.util.List;

import javax.xml.transform.OutputKeys;
```

```
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.stream.StreamResult;

import org.xml.sax.InputSource;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.StaxDriver;

public class XStreamTester {
   public static void main(String args[]){
      XStreamTester tester = new XStreamTester();
      XStream xstream = new XStream(new StaxDriver());
      Student student = tester.getStudentDetails();
      //Object to XML Conversion
      String xml = xstream.toXML(student);
      System.out.println(formatXml(xml));
   }

   private Student getStudentDetails(){
      Student student = new Student("Mahesh");
      student.addNote(new Note("first","My first assignment."));
      student.addNote(new Note("second","My Second assignment."));
      return student;
   }

   public static String formatXml(String xml){
      try{
         Transformer serializer= SAXTransformerFactory.newInstance(
         serializer.setOutputProperty(OutputKeys.INDENT, "yes");
         serializer.setOutputProperty("{http://xml.apache.org/xslt]
         Source xmlSource=new SAXSource(new InputSource(new ByteArr
         StreamResult res =  new StreamResult(new ByteArrayOutputSt
         serializer.transform(xmlSource, res);
         return new String(((ByteArrayOutputStream)res.getOutputStr
      }catch(Exception e){
         return xml;
      }
   }
}

class Student {
   private String studentName;
   private List<Note> notes = new ArrayList<Note>();
   public Student(String name) {
      this.studentName = name;
   }
   public void addNote(Note note) {
      notes.add(note);
   }
   public String getName(){
```

```
      return studentName;
   }
   public List<Note> getNotes(){
      return notes;
   }
}

class Note {
   private String title;
   private String description;
   public Note(String title, String description) {
      this.title = title;
      this.description = description;
   }
   public String getTitle(){
      return title;
   }
   public String getDescription(){
      return description;
   }
}
```

验证结果

使用javac编译器编译如下类：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>javac XStreamTester.java
```

现在运行XStreamTester看到的结果：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>java XStreamTester
```

验证输出

```
<?xml version="1.0" encoding="UTF-8"?>
<com.yiibai.xstream.Student>
  <studentName>Mahesh</studentName>
  <notes>
    <com.yiibai.xstream.Note>
      <title>first</title>
      <description>My first assignment.</description>
    </com.yiibai.xstream.Note>
    <com.yiibai.xstream.Note>
      <title>second</title>
      <description>My Second assignment.</description>
    </com.yiibai.xstream.Note>
  </notes>
</com.yiibai.xstream.Student>
```

在上面的结果，我们已经看到了Student对象名称是完全合格的。要替换它作为学生的标签，按照下面的链接。

类混叠

另外，在上述结果中可以看出，所需studentName要重命名来命名。要取代它，按照下面的链接。

字段混叠

在上面的结果，我们可以看到，笔记标记被添加成为笔记列表。替换它，按照下面的链接。

隐式集合混叠

在上面的结果，我们可以看到这个名字来作为一个子节点，需要将它作为根节点的属性。替换它，按照下面的链接。

属性混叠 包混叠

# XStream注解 - XStream教程

XStream支持注释做同样的任务，正如我们在前面的章节中是自动配置，而不是代码1所做的。在前面的章节中，我们已经看到了下面的代码配置。

```
xstream.alias("student", Student.class);
xstream.alias("note", Note.class);
xstream.useAttributeFor(Student.class, "studentName");
xstream.aliasField("name", Student.class, "studentName");
xstream.addImplicitCollection(Student.class, "notes");
```

下面的代码片段来说明使用annoations更容易的方式做同样的工作。

```
@XStreamAlias("student")    //define class level alias
class Student {

    @XStreamAlias("name")   //define field level alias
    @XStreamAsAttribute     //define field as attribute
    private String studentName;

    @XStreamImplicit        //define list as an implicit collectior
    private List<Note> notes = new ArrayList<Note>();

    @XStreamOmitField        //omit a field to not to be a part of
    private int type;
}
```

让我们使用XStream测试上面的注释。

创建一个Java类名为XStreamTester文件在
C:\>XStream_WORKSPACE\com\yiibai\xstream.

*File: XStreamTester.java*

```
package com.yiibai.xstream;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.util.ArrayList;
import java.util.List;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
```

```
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.stream.StreamResult;

import org.xml.sax.InputSource;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.annotations.XStreamAsAttribute;
import com.thoughtworks.xstream.annotations.XStreamImplicit;
import com.thoughtworks.xstream.annotations.XStreamOmitField;
import com.thoughtworks.xstream.io.xml.StaxDriver;

public class XStreamTester {
   public static void main(String args[]){
      XStreamTester tester = new XStreamTester();
      XStream xstream = new XStream(new StaxDriver());
      Student student = tester.getStudentDetails();
      xstream.processAnnotations(Student.class);

      //Object to XML Conversion
      String xml = xstream.toXML(student);
      System.out.println(formatXml(xml));
   }

   private Student getStudentDetails(){
      Student student = new Student("Mahesh");
      student.addNote(new Note("first","My first assignment."));
      student.addNote(new Note("second","My Second assignment."));
      student.setType(1);
      return student;
   }

   public static String formatXml(String xml){
      try{
         Transformer serializer= SAXTransformerFactory.newInstance(
         serializer.setOutputProperty(OutputKeys.INDENT, "yes");
         serializer.setOutputProperty("{http://xml.apache.org/xslt]
         Source xmlSource=new SAXSource(new InputSource(new ByteArr
         StreamResult res =  new StreamResult(new ByteArrayOutputSt
         serializer.transform(xmlSource, res);
         return new String(((ByteArrayOutputStream)res.getOutputStr
      }catch(Exception e){
         return xml;
      }
   }
}

@XStreamAlias("student")
class Student {

   @XStreamAlias("name")
   @XStreamAsAttribute
```

```
   private String studentName;

   @XStreamImplicit
   private List<Note> notes = new ArrayList<Note>();

   public Student(String name) {
      this.studentName = name;
   }

   public void addNote(Note note) {
      notes.add(note);
   }

   public String getName(){
      return studentName;
   }

   public List<Note> getNotes(){
      return notes;
   }

   @XStreamOmitField
   private int type;

   public int getType(){
      return type;
   }

   public void setType(int type){
      this.type = type;
   }
}

@XStreamAlias("note")
class Note {
   private String title;
   private String description;

   public Note(String title, String description) {
      this.title = title;
      this.description = description;
   }

   public String getTitle(){
      return title;
   }

   public String getDescription(){
      return description;
   }
}
```

验证结果

使用javac编译器编译如下类：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>javac XStreamTester.java
```

现在运行XStreamTester看到的结果：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>java XStreamTester
```

验证结果

```
<?xml version="1.0" encoding="UTF-8"?>
<student name="Mahesh">
  <note>
    <title>first</title>
    <description>My first assignment.</description>
  </note>
  <note>
    <title>second</title>
    <description>My Second assignment.</description>
  </note>
</student>
```

为了告诉XStream框架来处理注释，需要XML序列化之前添加下面的命令。

```
xstream.processAnnotations(Student.class);
```

或者

```
xstream.autodetectAnnotations(true);
```

# XStream对象流 - XStream教程

XStream提供java.io.ObjectInputStream和java.io.ObjectOutputStream替代实现，使对象流可以被序列化或XML序列化。当大对象集要被处理，保持在存储器中的一个对象，这是特别有用的。

## 语法：createObjectOutputStream()

```
ObjectOutputStream objectOutputStream = xstream.createObjectOutputS
```

## 语法 :createObjectInputStream()

```
ObjectInputStream objectInputStream = xstream.createObjectInputStre
```

现在，让我们用XStream对象流测试代码。

创建一个Java类名为XStreamTester文件在 C:\>XStream_WORKSPACE\com\yiibai\xstream.

*File: XStreamTester.java*

```
package com.yiibai.xstream;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.io.xml.StaxDriver;

public class XStreamTester {
   public static void main(String args[]){
      XStreamTester tester = new XStreamTester();
      XStream xstream = new XStream(new StaxDriver());
      xstream.autodetectAnnotations(true);

      Student student1 = new Student("Mahesh","Parashar");
      Student student2 = new Student("Suresh","Kalra");
```

```
      Student student3 = new Student("Ramesh","Kumar");
      Student student4 = new Student("Naresh","Sharma");

      try {
         ObjectOutputStream objectOutputStream = xstream.createObje
         objectOutputStream.writeObject(student1);
         objectOutputStream.writeObject(student2);
         objectOutputStream.writeObject(student3);
         objectOutputStream.writeObject(student4);
         objectOutputStream.writeObject("Hello World");
         objectOutputStream.close();

         ObjectInputStream objectInputStream = xstream.createObject
         Student student5 = (Student)objectInputStream.readObject(
         Student student6 = (Student)objectInputStream.readObject(
         Student student7 = (Student)objectInputStream.readObject(
         Student student8 = (Student)objectInputStream.readObject(
         String text = (String)objectInputStream.readObject();
         System.out.println(student5);
         System.out.println(student6);
         System.out.println(student7);
         System.out.println(student8);
         System.out.println(text);

      } catch (IOException e) {
         e.printStackTrace();
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
   }
}

@XStreamAlias("student")
class Student {

   private String firstName;
   private String lastName;

   public Student(String firstName, String lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

   public String getFirstName() {
      return firstName;
   }

   public String getLastName() {
      return lastName;
   }

   public String toString(){
      return "Student [ firstName: "+firstName+", lastName: "+ last
```

```
    }
}
```

验证结果

使用javac编译器编译如下类：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>javac XStreamTester.java
```

现在运行XStreamTester看到的结果：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>java XStreamTester
```

验证输出

```
Student [ firstName: Mahesh, lastName: Parashar ]
Student [ firstName: Suresh, lastName: Kalra ]
Student [ firstName: Ramesh, lastName: Kumar ]
Student [ firstName: Naresh, lastName: Sharma ]
Hello World
```

查看test.txt的内容在C:\>XStream_WORKSPACE\com\yiibai\xstream 文件夹。

```
<?xml version="1.0" ?>
<object-stream>
   <student>
      <firstName>Mahesh</firstName>
      <lastName>Parashar</lastName>
   </student>
   <student>
      <firstName>Suresh</firstName>
      <lastName>Kalra</lastName>
   </student>
   <student>
      <firstName>Ramesh</firstName>
      <lastName>Kumar</lastName>
   </student>
   <student>
      <firstName>Naresh</firstName>
      <lastName>Sharma</lastName>
   </student>
   <string>Hello World</string>
</object-stream>
```

# XStream对象流 - XStream教程

XStream提供java.io.ObjectInputStream和java.io.ObjectOutputStream替代实现，使对象流可以被序列化或XML序列化。当大对象集要被处理，保持在存储器中的一个对象，这是特别有用的。

## 语法：createObjectOutputStream()

```
ObjectOutputStream objectOutputStream = xstream.createObjectOutputS
```

## 语法 :createObjectInputStream()

```
ObjectInputStream objectInputStream = xstream.createObjectInputStre
```

现在，让我们用XStream对象流测试代码。

创建一个Java类名为XStreamTester文件在C:\>XStream_WORKSPACE\com\yiibai\xstream.

*File: XStreamTester.java*

```
package com.yiibai.xstream;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.io.xml.StaxDriver;

public class XStreamTester {
   public static void main(String args[]){
      XStreamTester tester = new XStreamTester();
      XStream xstream = new XStream(new StaxDriver());
      xstream.autodetectAnnotations(true);

      Student student1 = new Student("Mahesh","Parashar");
      Student student2 = new Student("Suresh","Kalra");
```

```
      Student student3 = new Student("Ramesh","Kumar");
      Student student4 = new Student("Naresh","Sharma");

      try {
         ObjectOutputStream objectOutputStream = xstream.createObje
         objectOutputStream.writeObject(student1);
         objectOutputStream.writeObject(student2);
         objectOutputStream.writeObject(student3);
         objectOutputStream.writeObject(student4);
         objectOutputStream.writeObject("Hello World");
         objectOutputStream.close();

         ObjectInputStream objectInputStream = xstream.createObject
         Student student5 = (Student)objectInputStream.readObject(
         Student student6 = (Student)objectInputStream.readObject(
         Student student7 = (Student)objectInputStream.readObject(
         Student student8 = (Student)objectInputStream.readObject(
         String text = (String)objectInputStream.readObject();
         System.out.println(student5);
         System.out.println(student6);
         System.out.println(student7);
         System.out.println(student8);
         System.out.println(text);

      } catch (IOException e) {
         e.printStackTrace();
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
   }
}

@XStreamAlias("student")
class Student {

   private String firstName;
   private String lastName;

   public Student(String firstName, String lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

   public String getFirstName() {
      return firstName;
   }

   public String getLastName() {
      return lastName;
   }

   public String toString(){
      return "Student [ firstName: "+firstName+", lastName: "+ last
```

```
    }
}
```

验证结果

使用javac编译器编译如下类：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>javac XStreamTester.java
```

现在运行XStreamTester看到的结果：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>java XStreamTester
```

验证输出

```
Student [ firstName: Mahesh, lastName: Parashar ]
Student [ firstName: Suresh, lastName: Kalra ]
Student [ firstName: Ramesh, lastName: Kumar ]
Student [ firstName: Naresh, lastName: Sharma ]
Hello World
```

查看test.txt的内容在C:\>XStream_WORKSPACE\com\yiibai\xstream 文件夹。

```
<?xml version="1.0" ?>
<object-stream>
   <student>
      <firstName>Mahesh</firstName>
      <lastName>Parashar</lastName>
   </student>
   <student>
      <firstName>Suresh</firstName>
      <lastName>Kalra</lastName>
   </student>
   <student>
      <firstName>Ramesh</firstName>
      <lastName>Kumar</lastName>
   </student>
   <student>
      <firstName>Naresh</firstName>
      <lastName>Sharma</lastName>
   </student>
   <string>Hello World</string>
</object-stream>
```

# XStream编写JSON - XStream教程

XStream支持JSON通过初始化XStream对象适当的驱动程序。 XStream目前支持JettisonMappedXmlDriver和JsonHierarchicalStreamDriver。

现在，让我们使用XStream处理JSON的代码测试。

创建一个Java类名为XStreamTester文件在C:\>XStream_WORKSPACE\com\yiibai\xstream.

*File: XStreamTester.java*

```
package com.yiibai.xstream;

import java.io.Writer;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.io.HierarchicalStreamWriter;
import com.thoughtworks.xstream.io.json.JsonHierarchicalStreamDrive
import com.thoughtworks.xstream.io.json.JsonWriter;

public class XStreamTester {
    public static void main(String args[]){
        XStreamTester tester = new XStreamTester();
         XStream xstream = new XStream(new JsonHierarchicalStreamDr
            public HierarchicalStreamWriter createWriter(Writer wri
                return new JsonWriter(writer, JsonWriter.DROP_ROOT_
            }

        });

        Student student = new Student("Mahesh","Parashar");

        xstream.setMode(XStream.NO_REFERENCES);
        xstream.alias("student", Student.class);

        System.out.println(xstream.toXML(student));

    }
}

@XStreamAlias("student")
class Student {

    private String firstName;
    private String lastName;

    public Student(String firstName, String lastName) {
        this.firstName = firstName;
```

```
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String toString(){
        return "Student [ firstName: "+firstName+", lastName: "+ la
    }
}
```

验证结果

使用javac编译器编译如下类：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>javac XStreamTester.java
```

现在运行XStreamTester看到的结果：

```
C:\XStream_WORKSPACE\com\yiibai\xstream>java XStreamTester
```

验证输出

```
{
  "firstName": "Mahesh",
  "lastName": "Parashar"
}
```